

全国计算机技术与软件专业技术资格（水平）考试参考用书

软件设计师考试同步辅导

（下午科目）

工业和信息化部教育与考试中心　推荐

谢瑜 周胜 主编 / 鲁磊纪 杨章静 副主编

第4版

清华大学出版社

全国计算机技术与软件专业技术资格(水平)考试参考用书

软件设计师考试同步辅导

(下午科目)(第4版)

谢　瑜　周　胜　主　编

鲁磊纪　杨章静　副主编

清华大学出版社

北　京

内 容 简 介

本书是按照人力资源和社会保障部、工业和信息化部2009年颁布的全国计算机技术与软件专业技术资格(水平)考试大纲和指定教材编写的考试用书。全书共分为7章，内容包括数据流图设计、数据库设计、UML分析与设计、程序流程图、算法设计、面向对象程序设计和样卷模拟，主要从考试大纲要求、考点辅导、典型例题分析、同步练习和本章小结几个方面对各部分内容加以系统地阐释。

本书具有考点分析透彻、例题典型、习题丰富等特点，非常适合备考软件设计师的考生使用，也可作为高等院校或培训班的教材。

图书在版编目(CIP)数据

软件设计师考试同步辅导(下午科目)/谢瑜，周胜主编. —4版. —北京：清华大学出版社，2018
(全国计算机技术与软件专业技术资格(水平)考试参考用书)
ISBN 978-7-302-50550-1

Ⅰ. ①软… Ⅱ. ①谢… ②周… Ⅲ. ①软件设计—资格考试—自学参考资料 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2018)第145344号

责任编辑：魏 莹 李玉萍
封面设计：常雪影
责任校对：李玉茹
责任印制：丛怀宇
出版发行：清华大学出版社
网 址：http://www.tup.com.cn, http://www.wqbook.com
地 址：北京清华大学学研大厦A座 **邮 编：**100084
社 总 机：010-62770175 **邮 购：**010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 装 者：清华大学印刷厂
经 销：全国新华书店
开 本：185mm×260mm **印 张：**29 **插 页：**2 **字 数：**702千字
版 次：2005年6月第1版 2018年8月第4版 **印 次：**2018年8月第1次印刷
定 价：86.00元

产品编号：071203-01

前　言

全国计算机技术与软件专业技术资格(水平)考试自实施起至今已经历了 20 多年，在社会上产生了很大的影响，其权威性得到社会各界的广泛认可。为适应我国信息化发展的需求，国家人力资源和社会保障部同工业和信息化部在 2009 年对软件设计师级别考试大纲进行了重新调整，以满足社会上对各种信息技术人才的需要。本书第 1 版自 2005 年、第 2 版自 2010 年、第 3 版自 2013 年出版以来，被众多考生选用为考试参考书，多次重印，深受广大读者好评。为了与考试同步，本书对第 3 版同名书进行了更新升级，将最新考试真题穿插其中。更新升级后本书特色如下。

(1) 知识点全面。本书与 2009 年软件设计师考试大纲考试科目 2——软件设计基本一致，又兼顾计算机技术发展和知识更新，对属于大纲要求的知识点而指定教材没有阐述的部分进行了必要的补充。

(2) 结构与官方教程同步。本书参考最新指定官方教程(2018 年出版)、最新考试大纲及最新题型编写章节内容，便于考生使用《软件设计师教程(第 5 版)》同步复习，同时更加突出重点与难点，针对性强，减轻考生复习的压力。

(3) 例题与习题经典。最近 4 年(2014—2017 年)8 次考试真题全部被分类解析到例题中，并同时在其中增加了根据最新考试大纲精心设计的例题，具有典型性和代表性，而 2013 年 2 次考试真题被分类归入同步练习中，使考生能从以前的考题中更好地熟悉考试的难度与广度，顺利通过考试。

(4) 重点突出。第 4 版沿袭前一版的框架，每一小节分 4 个模块：考点辅导、典型例题分析、同步练习和同步练习参考答案。其中，考点辅导部分主要以专题的方式，重点介绍软件设计师下午考试所需的各个方面的知识；典型例题分析是本书的重点，它详尽细致地剖析了最近 4 年(2014—2017 年)的真题和例题；同步练习每一道题都配有标准的答案。

(5) 语言精练。对语言进行了锤炼，概念更准确、更清晰，覆盖所有大纲考点，并突出重点和难点。

(6) 例题全面。对书中所有例题与习题进行了精选，确保所有题目符合考纲要求，例题选取典型、有梯度、有广度，分析详尽；题目的难易度、分布率与真实考试相当；题目答案正确、解析科学；无重复、雷同题目。

本书非常适合备考软件设计师的考生使用，也可作为高等学校相关专业或培训班的教材。

本书由谢瑜、周胜担任主编，鲁磊纪、杨章静担任副主编，参与本书组织、编写和资料收集的还有王华君、陶佳、史国川、徐国明、刁爱军、陈海峰、赵晗、吴敏、刘立军、

宋白玉、石鲁生、何光明等。在此对本书第 1 版、第 2 版、第 3 版的作者及全体参与人员表示衷心的感谢。本书在编写的过程中，参考了许多相关的书籍和资料，从中汲取了许多营养，在此也对这些参考文献的作者表示感谢。需要特别提出感谢的是来自互联网的各位不知姓名的网友们的无私奉献，正是由于你们，才使本书的内容更完善、更详尽。

由于作者水平所限，书中难免存在错漏和不妥之处，敬请读者批评指正。联系邮箱：iteditor@126.com。

编　者

软件设计师考试(下午)考点分布导航图

章	历年真题分布								大纲解读	阅读链接	命题预测
	2014.05	2014.11	2015.05	2015.11	2016.05	2016.11	2017.05	2017.11			
第 1 章 数据流图设计	补充外部实体和数据存储名称，补充加工处理及绘制数据流时要注意的问题，找出错误数据流(共 15 分)	补充外部实体和数据存储名称、数据流的名称及其起点和终点(共 15 分)	补充外部实体和数据存储名称，补充加工处理、数据流的名称及其起点和终点(共 15 分)	补充外部实体和数据存储名称，补充加工处理、数据流的名称及其起点和终点、补充加工处理及绘制数据流时要注意的问题(共 15 分)	补充外部实体和数据存储名称、数据流的名称及其起点和终点，补充加工处理及绘制数据流时要注意的问题(共 15 分)	补充外部实体和数据存储名称，补充加工处理、数据流的名称及其起点和终点，补充加工处理及绘制数据流时要注意的问题(共 15 分)	补充外部实体和数据存储名称，补充加工处理、数据流的名称及其起点和终点(共 15 分)	补充外部实体和数据存储名称，补充加工处理、数据流的名称及其起点和终点，补充加工处理及绘制数据流时要注意的问题(共 15 分)	**未变动的大纲要求** ①理解系统需求说明。 ②制定详细的工作流和数据流。 ③理解和掌握数据流图的基本概念，包括逻辑数据流图和物理数据流图的区别和联系。 ④理解各子系统和上下层数据流图的关系，掌握数据流图的原则和规律。 ⑤了解用于系统设计的转换图、状态迁移图等	**1. 考题分布** 通常出现在软件设计师考试(下午科目)的第 1 题。 **2. 阅读建议** 本章对应《软件设计师教程(第 3 版)(修订版)》(以下简称“教程”)中 12.1 节“结构化分析与设计”的内容。考生可以对照教程相应部分进行同步复习	本章考点分值约占总考分的 20%。高频考点为: ◆找出遗漏数据流; ◆指出错误/多余的数据流; ◆补充加工处理; ◆数据字典
第 2 章 数据库设计	E-R 图的画法、主键与外键的概念、实体联系与联系的类型(共 15 分)	E-R 图的画法、主键与外键的概念、实体联系与联系的类型 (共 15 分)	实体联系与联系的类型、关系模式的设计(共 15 分)	E-R 图的画法、实体联系与联系的类型、关系模式的设计(共 15 分)	E-R 图的画法、实体联系与联系的类型、主键与外键的概念(共 15 分)	E-R 图的画法、主键与外键的概念、逻辑结构设计(共 15 分)	实体联系与联系的类型、关系模式的设计(共 15 分)	E-R 图的画法、主键与外键的概念(共 15 分)	**未变动的大纲要求** ①设计 E-R 模型及其他数据模型。 ②设计关系模式。 ③数据库语言(SQL)。 ④数据库访问	**1. 考题分布** 通常出现在软件设计师考试(下午科目)的第 2 题。 **2. 阅读建议** 本章对应教程 12.2 节“数据库分析与设计”的内容。考生可以对照教程相应部分进行同步复习	本章考点分值约占总考分的 20%。高频考点为: ◆E-R 图的画法; ◆关系模式的设计; ◆数据库的概念结构设计和逻辑结构设计; ◆SQL 语言及索引相关知识

续表

章	历年真题分布								大纲解读	阅读链接	命题预测
	2014.05	2014.11	2015.05	2015.11	2016.05	2016.11	2017.05	2017.11			
第 3 章 UML 分析与设计	用例、类名、选设计模型(共 15 分)	用例、类名、属性、选设计模型 (共 15 分)	用例、类名、设计模型(共 15 分)	用例、类名、属性(共 15 分)	用例、类名、属性(共 15 分)	状态名、事件名、类名(共 15 分)	类名、用例、状态名 (共 15 分)	类名、属性、选设计模型(共 15 分)	**未变动的大纲要求** ①掌握面向对象的分析与设计方法。 ②掌握 UML 描述方法的相关知识	**1. 考题分布** 通常出现在软件设计师考试(下午科目)的第 3 题。 **2. 阅读提示** 本章对应教程 12.3 节“面向对象分析与设计”的内容。考生可以对照教程相应部分进行同步复习。 **3. 贴心提醒** UML 的各种视图是考试的重点，本书比教材对其的介绍更为详细，供考生有针对性复习	本章考点分值约占总考分的 20%。高频考点为: ◆类的属性及对象间的结构关系; ◆UML 用例图和活动图; ◆UML 用例之间的关系和内涵
第 4 章 程序流程图	近几年没有直接考查。								**未变动的大纲要求** 掌握程序算法的描述方法，着重掌握程序流程图这一方法	**1. 考题分布** 通常出现在软件设计师考试(下午科目)的第 4 题。 **2. 阅读建议** 本章内容在教程上没有相关内容。考生可完全参看本书复习。 **3. 贴心提醒** 程序流程图是算法描述方法的重要手段，本章内容在大纲上明确提出，故本书将其整理编排，补充为第 4 章程序流程图。着重介绍了程序流程图的相关知识	本章考点分值约占总考分的 8%。出题方式为: ◆用程序流程图的方法描述一个实际问题; ◆用程序流程图描述一种常见的算法
第 5 章 算法设计	归并排序、时间复杂度(15 分)	动态规划、时间复杂度、排序(共 15 分)	数组的应用、回溯算法(共15分)	动态规划、递归、时间复杂度(15 分)	动态规划、递归、时间复杂度(15 分)	数组的应用、时间复杂度 (共 15 分)	分治法、时间复杂度、(共 15 分)	回溯算法、最先适宜算法和最优适宜算法，时间复杂度(共 15 分)	**1. 未变动的大纲要求** ①掌握 C 程序设计语言。 ②掌握常用数据结构及其常用操作。 ③掌握排序算法、查找算法。 ④灵活应用各种算法设计策略。 **2. 删除的大纲要求** 指导程序员编程和测试，并进行必要的优化。 **3. 对大纲变化的说明** 最新的大纲要求只掌握 C 语言程序设计即可。对于删除的大纲内容，历年均未出现考题，甚至目前也无法具体地考这方面的内容，考生可不再复习该部分内容	**1. 考题分布** 通常出现在软件设计师考试(下午科目)的第4题或者第 5 题。 **2. 阅读提示** 本章对应教程 12.4 节“算法分析与设计”和 12.5 节“面向过程的程序设计与实现”的内容。考生可以对照教程相应内容进行同步复习。 **3. 贴心提醒** 除了介绍教程上的内容，本书针对常考内容，还着重介绍了几个常见的算法思想和设计，供考生更加有针对性地复习	本章考点分值约占总考分的 32%。高频考点为: ◆常见数据结构(链表、树)的操作; ◆实际问题与常见数据结构相结合的程序设计; ◆常见算法的程序设计

续表

章	历年真题分布								大纲解读	阅读链接	命题预测
	2014.05	2014.11	2015.05	2015.11	2016.05	2016.11	2017.05	2017.11			
第6章 面向对象程序设计	C++程序设计(类的用例和继承)(共15分)	C++程序设计(类的用例和继承)(共15分)	C++程序设计(类的用例)、访问者模式(共15分)	C++程序设计(抽象类、策略模式)(共15分)	C++程序设计(抽象类和多态)(共15分)	C++程序设计(装饰模式)(共15分)	C++程序设计(生成器模式)(共15分)	桥接设计模式下的C++程序设计(共15分)	**1. 未变动的大纲要求** 掌握C++、Java中任一种程序设计语言。 **2. 删除的大纲要求** ①掌握Visual Basic程序设计语言。 ②指导程序员编程和测试，并进行必要的优化。 **3. 对大纲变化的说明** 最新的大纲要求只掌握C++或Java中的一种语言即可，并且从近几年的出题情况来看，只要求程序填空，不需要测试和优化	**1. 考题分布** 通常出现在软件设计师考试(下午科目)的第6题和第7题。 **2. 阅读建议** 本章对应教程12.5节“面向对象的程序设计与实现”的内容。章节的结构安排与教程不同。本书根据考题的类型，介绍两种不同的程序设计语言，而教程则是以两种程序设计语言来展现面向对象程序设计的实现过程。建议考生以本书复习为主，如果缺乏面向对象程序设计的实现过程的考生可同时参看教程	本章考点分值约占总考分的40%。但是通常放在选做题部分，考生只需选择一道自己擅长的程序设计语言题目答题即可。高频考点为： ◆结合一些设计模式进行C++程序设计； ◆结合一些设计模式进行Java程序设计
	Java程序设计(类的用例和继承)(共15分)	Java程序设计(类的用例和继承)(共15分)	Java程序设计(类的用例)、访问者模式(共15分)	Java程序设计(抽象类、策略模式)(共15分)	Java程序设计(抽象类和多态)(共15分)	Java程序设计(装饰模式)(共15分)	Java程序设计(生成器模式)(共15分)	桥接设计模式下的Java程序设计(共15分)			

目　　录

第 1 章　数据流图设计

大纲要求：

- 理解和掌握数据流图的基本概念，包括逻辑数据流图和物理数据流图的区别和联系。
- 理解系统需求说明，根据需求说明绘制出数据流图，设计系统数据流的输入/输出。
- 理解各子系统和上下层数据流图的关系，掌握数据流图的原则和规律。
- 了解用于系统设计的转换图、状态迁移图等。

1.1　数据流图设计的基础知识

1.1.1　考点辅导

根据考纲要求以及近几年软件设计师水平考试试题分布情况来看，数据流图的设计已经成为必考的知识点。数据流图本身的特点使得考查的题型比较集中，常出的考题类型有：找出遗漏的数据流，指出错误的数据流，找出多余的数据流，找出数据流图中的多余文件。近几年把数据字典、数据库、面向对象程序设计等知识也结合到了数据流图中考查，但难度都不大。所以，数据流图是拿分的题型，考生一定要好好把握，多做练习，熟悉解题方法，掌握解题技巧。

解答数据流图的题目关键在于细心。考试时一定要仔细阅读题目说明和给出的流程图。另外，解题时要懂得将说明和流程图进行对照，将父图和子图进行对照，切忌按照常识来猜测。同时应按照一定顺序考虑问题，以防遗漏，比如可以按说明的顺序，或是按数据流向的顺序逐个排除和分析。

下面就一些常见的题型作一下解题分析。

1.1.1.1　数据流图的基本概念

数据流图的考查中需要考生掌握数据流图的基本概念，另外，还会涉及数据字典、数据库、面向对象方法、转换图、状态迁移图等概念，考生对这些概念都要非常清晰。

对于基本概念的考查一般结合在题目中，有时也会针对这些基本概念出题，比如有的题目要求说明逻辑数据流图和物理数据流图之间的主要区别。

1.　基本概念

数据流图又称数据流程图(Data Flow Diagram，DFD)，是一种便于用户理解、分析系统数据流程的图形工具。它摆脱了系统的物理内容，精确地在逻辑上描述系统的功能、输入、输出和数据存储等，是系统逻辑模型的重要组成部分。

2.　DFD 的基本成分

DFD 的基本成分及其图形表示方法如图 1-1 所示。

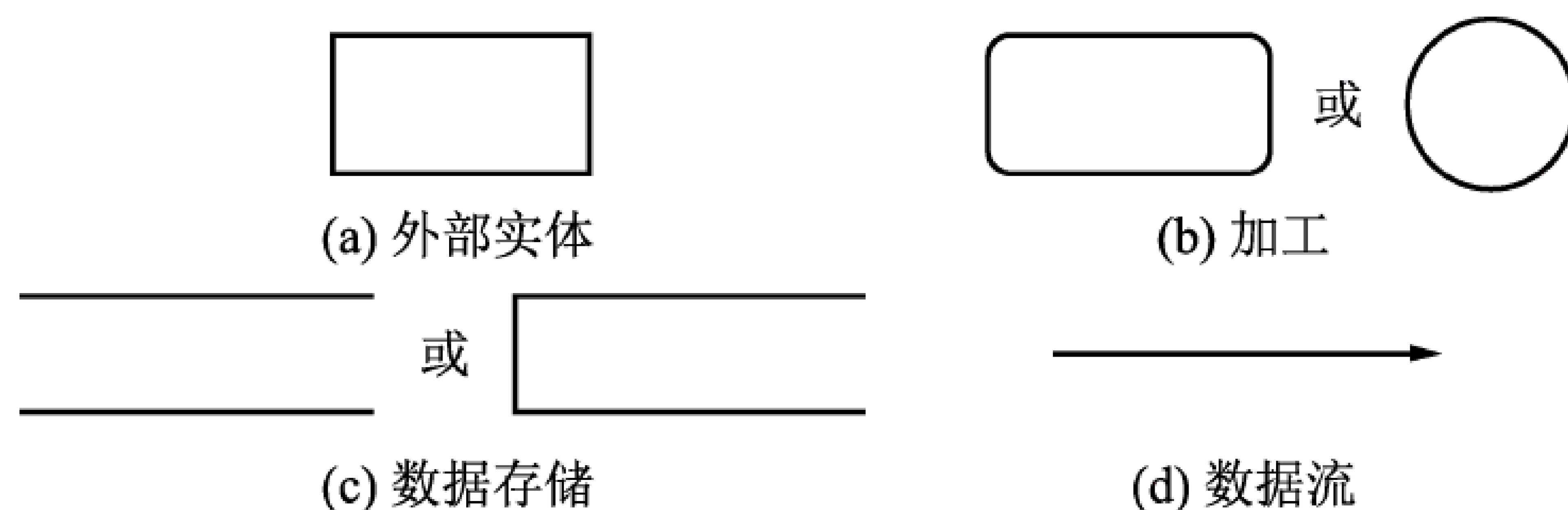


图 1-1　DFD 的基本成分及图形表示方法

(1) 外部实体(External Agent)。外部实体是指存在于软件系统之外的人员或组织，它指出系统所需数据的发源地和系统所产生的数据的归宿地。

(2) 加工(Process)。加工描述了输入数据流到输出数据流之间的变换，也就是输入数据流经过什么处理后变成了输出数据流。每个加工都有一个名字和编号。编号能反映出该加工位于分层 DFD 中的哪个层次和哪张图中，也能够看出它是哪个加工分解出来的子加工。

(3) 数据存储(Data Store)。数据存储用来表示存储的数据，每个数据存储都有一个名字。

(4) 数据流(Data Flow)。数据流由一组固定成分的数据组成，表示数据的流向。值得注意的是，DFD 中描述的是数据流，而不是控制流。除了流向数据存储或从数据存储流出的数据流不必命名外，每个数据流都必须有一个合适的名字，以反映该数据流的含义。

3. 分层数据流图的画法

(1) 画系统的输入和输出。把整个软件系统看作一个大的加工，然后根据系统从哪些外部实体接收数据流，以及系统发送数据流到哪些外部实体，就可以画出系统的输入和输出图，这张图称为顶层图。

(2) 画系统的内部。将顶层图的加工分解成若干个加工，并用数据流将这些加工连接起来，使得顶层图中的输入数据经过若干个加工处理后变换成顶层图的输出数据流。这张图称为 0 层图。从一个加工画出一张数据流图的过程实际上就是对这个加工的分解。

可以用下述方法来确定加工：在数据流的组成或值发生变化的地方画一个加工，这个加工的功能就是实现这一变化；也可根据系统的功能确定加工。

确定数据流的方法：当用户把若干个数据看作一个单位来处理(这些数据一起到达，一起加工)时，可把这些数据看成一个数据流。

对于一些以后某个时间要使用的数据，可以组织成一个数据存储来表示。

(3) 画加工的内部。把每个加工看作一个小系统，该加工的输入/输出数据流看成小系统的输入/输出数据流。于是可以用与画 0 层图同样的方法画出每个加工的 DFD 子图。

对第(3)步分解出来的 DFD 子图中的每个加工重复第(3)步的分解，直至图中尚未分解的加工都足够简单(也就是说这种加工不必再分解)为止。至此，就得到了一套分层数据流图。

4. 对图和加工进行编号

对于一个软件系统，其数据流图可能有许多层，每一层又有许多张图。为了区分不同的加工和不同的 DFD 子图，应该对每张图和每个加工进行编号，以利于管理。

1) 父图与子图

假设分层数据流图里的某张图(记为图 A)中的某个加工可用另一张图(记为图 B)来分解，则称图 A 是图 B 的父图，图 B 是图 A 的子图。在一张图中，有些加工需要进一步分解，

有些加工则不必分解。因此，如果父图中有 n 个加工，那么它可以有 0~n 张子图(这些子图位于同一层)，但每张子图都只对应于一张父图。

2) 编号

顶层图只有一张，图中的加工也只有一个，所以不必编号。

0 层图只有一张，图中的加工号可以分别是 0.1, 0.2…或 1, 2…

子图号就是父图中被分解的加工号。

图的加工号由图号、圆点和序号组成。

5. 应注意的问题

(1) 应适当地为数据流、加工、数据存储、外部实体命名，且名字应反映该成分的实际含义，避免空洞的名字。

(2) 画数据流而不要画控制流。

(3) 每条数据流的输入或者输出都是加工。

(4) 一个加工的输出数据流不应与输入数据流同名，即使它们的组成成分相同。

(5) 允许一个加工有多条数据流流向另一个加工，也允许一个加工有两个相同的输出数据流流向两个不同的加工。

(6) 保持父图与子图平衡。也就是说，父图中某加工的输入/输出数据流必须与其子图的输入/输出数据流在数量和名字上相同。值得注意的是，如果父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流，而子图中组成这些数据流的数据项全体正好是父图中的这一个数据流，那么它们仍然算是平衡的。

(7) 在自顶向下的分解过程中，若一个数据存储首次出现时只与一个加工有关，那么这个数据存储应作为这个加工的内部文件而不必画出。

(8) 保持数据守恒。也就是说，一个加工所有输出数据流中的数据必须能从该加工的输入数据流中直接获得，或者是通过该加工产生的数据。

(9) 每个加工必须既有输入数据流，又有输出数据流。

(10) 在整套数据流图中，每个数据存储必须既有读的数据流，又有写的数据流。但在某一张子图中可能是只有读没有写，或者是只有写没有读。

1.1.1.2 补充和完善数据流

补充和完善数据流是数据流图最常出的题型，也是重点和难点。解答此类问题有一定的技巧，以一些常规的入口作为突破口，往往能事半功倍。

遇到这类问题，首先要想到分层数据流图的数据流平衡原则，即父图和子图的输入/输出数据流一致，这是找出遗漏数据流非常重要的技巧。其次，每个加工至少有一个输入数据流和一个输出数据流，反映此加工的数据来源和结果，加工的输出数据流应该都有其对应的输入数据流。再次，要找出遗漏的数据流，最根本的依据还是说明。因为除了图之外，题目中最重要的部分就是说明。说明部分详细介绍了系统的功能，是找出所缺数据流的基本入口。

有时数据流平衡原则不作为解题的直接方法，而作为排除的手段，然后根据说明或其他方法找到图中遗漏的数据流。

1.1.1.3　找出错误或多余的数据流

要找出错误或多余的数据流，解题方法可以参考完善数据流的方法。一般可以先进行上下层图的对照和分析，然后检查是否每个加工至少有一个输入数据流和一个输出数据流，是否加工的输出数据流都有其对应的输入数据流。而最根本的判断标准仍然是题目的说明部分。所以考生一定要耐心、认真地阅读题目中对系统功能的阐述和说明，然后解题时再次阅读说明，从中找到依据和突破口。

1.1.1.4　找出多余的文件

在某层数据流图中，只画流程图各加工之间的公共数据文件时，如果一个文件仅作用于一个加工，即和该文件有关的输入/输出数据流只涉及一个加工，那么该文件可以作为局部文件出现在该加工的子图中，在父图中则可以省略。这个规则是为了使整个流程图的层次结构更为清晰、科学。当然这些文件如果画出，并不会造成理解的错误。

另外，如果某层图只有一层细化图，即该层图没有子图，则不存在局部文件和外部文件之分，其中涉及的任何文件都不作为多余的文件。

1.1.1.5　添加数据字典条目

此类题一般难度比较小，可以根据说明部分找出答案。同时还可以结合给出的数据流图，查看有关记录需要输入给哪些加工，这些加工输出哪些字段。

1.1.2　典型例题分析

例 1　某会议中心提供举办会议的场地设施和各种设备，供公司与各类组织机构租用。场地包括一个大型报告厅、一个小型报告厅以及诸多会议室。这些报告厅和会议室可提供的设备有投影仪、白板、视频播放/回放设备、计算机等。为了加强管理，该中心欲开发一会议预订系统，系统的主要功能如下。(2016 年 5 月试题一)

(1) 检查可用性。客户提交预订请求后，检查预订表，判定所申请的场地是否在申请日期内可用；如果不可用，返回不可用信息。

(2) 临时预订。会议中心管理员收到客户预订请求的通知之后，提交确认。系统生成新临时预订存入预订表，并对新客户创建一条客户信息记录加以保存。根据客户记录给客户发送临时预订确认信息和支付定金要求。

(3) 分配设施与设备。根据临时预订或变更预订的设备和设施需求，分配所需设备(均能满足用户要求)和设施，更新相应的表和预订表。

(4) 确认预订。管理员收到客户支付定金的通知后，检查确认，更新预订表，根据客户记录给客户发送预订确认信息。

(5) 变更预订。客户还可以在支付余款前提交变更预订请求，对变更的预订请求检查可用性，如果可用，分配设施和设备；如果不可用，返回不可用信息。管理员确认变更后，根据客户记录给客户发送确认信息。

(6) 要求付款。管理员从预订表中查询距预订的会议时间两周内的预订，根据客户记录给满足条件的客户发送支付余款要求。

(7) 支付余款。管理员收到客户余款支付的通知后，检查确认，更新预订表中的已支付余款信息。

现采用结构化方法对会议预订系统进行分析与设计，获得如图 1-2 所示的上下文数据流图和图 1-3 所示的 0 层数据流图(不完整)。



图 1-2　上下文数据流图



图 1-3　0 层数据流图

【问题 1】(2 分)

使用说明中的词语，给出图 1-2 中的实体 E1~E2 的名称。

【问题 2】(4 分)

使用说明中的词语，给出图 1-3 中的数据存储 D1~D4 的名称。

【问题 3】(6 分)

根据说明和图中的术语，补充图 1-3 中缺失的数据流及其起点和终点。

【问题 4】(3 分)

如果发送给客户的确认信息是通过 E-mail 系统向客户信息中的电子邮件地址进行发送的，那么需要对图 1-2 和图 1-3 进行哪些修改？用 150 字以内文字加以说明。

解析：

该题以会议预订系统来考查学生对数据流图知识点的掌握程度。从题目的问答形式上来看和往年相似。

【问题 1】

根据 0 层数据流中 E1 向系统发送预订请求数据流可知，E1 实体为客户；从预订请求通知到临时预订确认可知 E2 实体为管理员。

【问题 2】

根据题目对功能的描述，结合 0 层数据流图，新临时预订提交、变更的预订请求提交等，可知 D1 为预订表；新客户信息存入 D2 中，可知 D2 为客户信息记录表；根据分配设施和设备数据流，可以得到 D3、D4 分别为设施表和设备表。

【问题 3】

由“确认预订”收到客户支付定金的通知后，检查确认更新预订表，同时要向客户发送预订确认信息，存在一个起点为 4(确认预订)到终点为 E1 的数据流，即预订确认信息数据流；根据临时预订描述，首先要由客户发送预订请求，提交确认，系统生成新临时预订存入预订表，所以存在一个起点为客户即 E1，终点为 2(临时预订)的数据流，即客户临时预订信息数据流。

【问题 4】略。

答案：

【问题 1】

E1：客户。

E2：管理员。

【问题 2】

D1：预订表。

D2：客户信息记录表。

D3：设施表。

D4：设备表。

【问题 3】

(1) 数据流名称：预订确认信息。起点：4(确认预订)。终点：E1。

(2) 数据流名称：客户信息。起点：E1。终点：2(临时预订)。

【问题 4】

图 1-2 中：增加外部实体“第三方 E-mail 系统”，将临时预订/预订/变更确认信息终点均修改至“第三方 E-mail 系统”。

图 1-3 中：增加外部实体“第三方 E-mail 系统”，增加加工“发送邮件”，将临时预订/预订/变更确认信息终点均修改至“发送邮件”加工，并增加从 D2 到“发送邮件”加工的数据流“电子邮件地址”，再从发送邮件加工引出数据流 临时预订/预订/变更确认信息，终点为“第三方 E-mail 系统”。

例 2　某慕课教育平台欲添加在线作业批改系统，以实现高效的作业提交与批改，并进行统计。学生和讲师的基本信息已经初始化为数据库中的学生表和讲师表。系统的主要功能如下。(2015 年 11 月试题一)

(1) 提交作业。验证学生标识后，学生将电子作业通过在线的方式提交，并进行存储。系统给学生发送通知表明提交成功，通知中包含唯一编号；并通知讲师有作业提交。

(2) 下载未批改作业。验证讲师标识后，讲师从系统中下载学生提交的作业。下载的作业将显示在屏幕上。

(3) 批改作业。讲师按格式为每个题目进行批改打分，并进行整体评价。

(4) 上传批改后的作业。将批改后的作业(包括分数和评价)返回给系统，进行存储。

(5) 记录分数和评价。将批改后的作业的分数和评价记录在学生信息中，并通知学生作业已批改过。

(6) 获取已批改作业。根据学生标识，给学生查看批改后的作业，包括提交的作业、分数和评价。

(7) 作业抽检。根据教务人员标识抽取批改后的作业样本，给出抽检意见，然后形成抽检报告给讲师。

现采用结构化方法对在线作业批改系统进行分析与设计，获得如图 1-4 所示的上下文数据流图和图 1-5 所示的 0 层数据流图。

【问题 1】(3 分)

使用说明中的词语，给出图 1-4 中的实体 E1~E3 的名称。

【问题 2】(4 分)

使用说明中的词语，给出图 1-5 中的数据存储 D1~D4 的名称。

【问题 3】(6 分)

根据说明和图中术语，补充图 1-5 中缺失的数据流及其起点和终点。

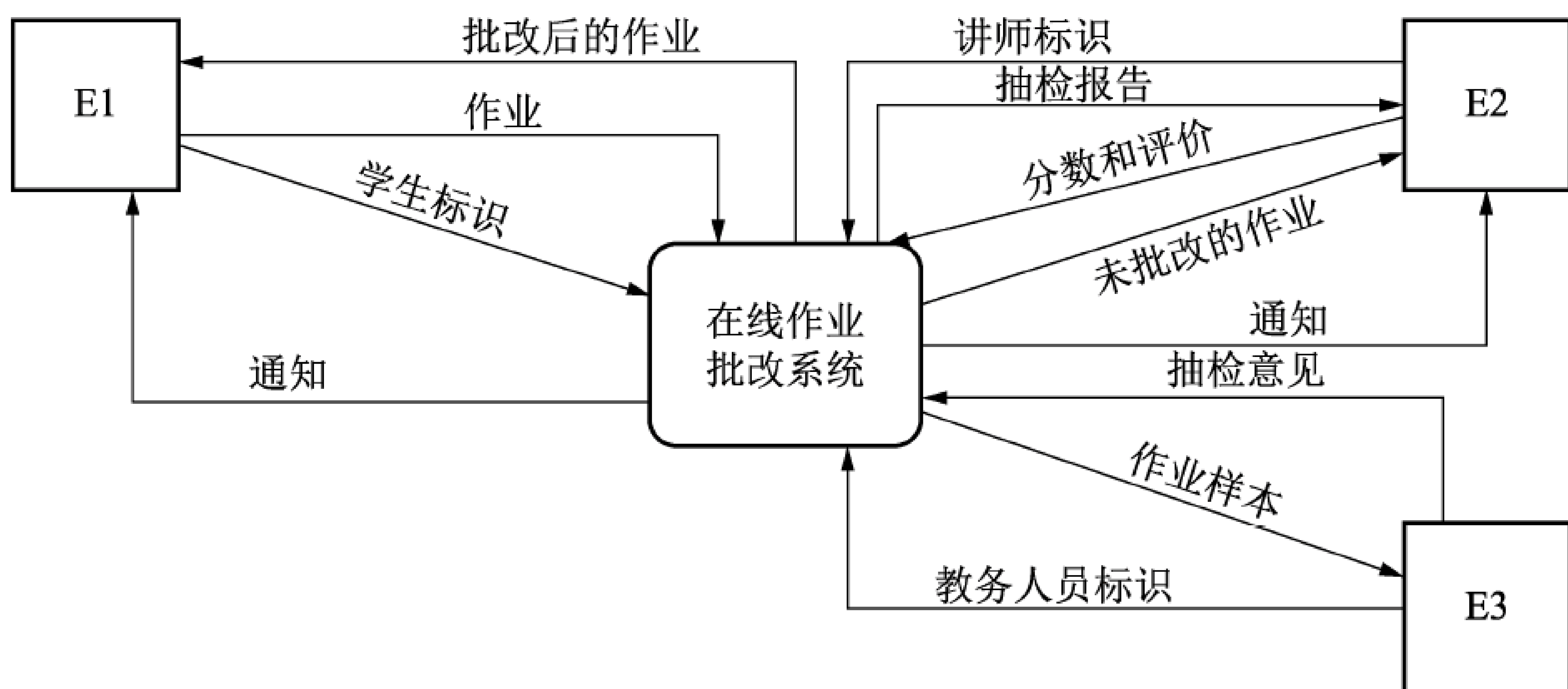


图 1-4　上下文数据流图

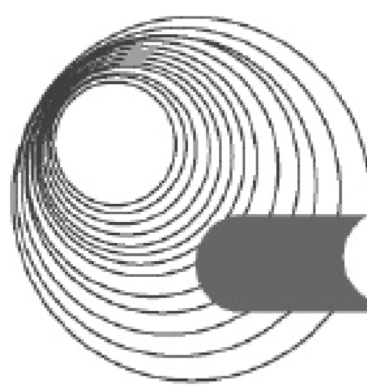


图 1-5　0 层数据流图

【问题 4】(2 分)

若发送给学生和讲师的通知是通过第三方 E-mail 系统进行的，则需要对图 1-4 和图 1-5 进行哪些修改？用 100 字以内文字加以说明。

解析：

【问题 1】

由系统功能说明知，学生将电子作业通过在线的方式提交系统，所以 E1 实体为“学生”；讲师从系统中下载学生提交的作业，进行批改打分并做整体评价，所以 E2 实体为“讲师”；教务人员抽取批改后的作业样本，给出抽检意见，所以 E3 实体为“教务人员”。

【问题 2】

学生和讲师的基本信息已经初始化为数据库中的学生表和讲师表，所以数据存储 D2、

D3 的名称分别为“学生表”和“讲师表”。学生将电子作业完成后通过在线的方式提交并存储，因此数据存储 D1 存储的是学生的作业，名称为“作业”；讲师批改完学生作业后，需将批改后的作业(包括分数和评价)返回给系统并进行存储，所以数据存储 D4 的名称为“批改后的作业”。

【问题 3】

学生提交作业之后，系统要给学生发送通知消息，显然数据流图中缺少这样一条数据流；讲师批改完作业后，将批改后的作业的分数和评价记录在学生信息，并通知学生作业批改过，这里缺少两条数据流；最后教务人员抽检后将结果报告讲师，这条数据流也缺少。

【问题 4】略。

答案：

【问题 1】

E1：学生。E2：讲师。E3：教务人员。

【问题 2】

D1：作业。D2：学生表。D3：讲师表。D4：批改后的作业。

【问题 3】

(1) 数据流名称：通知。起点：提交作业。终点：E1。

(2) 数据流名称：抽检报告。起点：作业抽检。终点：E2。

(3) 数据流名称：分数和评价。起点：记录分数和评价。终点：D2。

(4) 数据流名称：通知。起点：记录分数和评价。终点：E1。

【问题 4】

增加外部实体“第三方 E-mail 系统”，将原来的两条“通知”数据流合并为一条“通知”数据流，终点为“第三方 E-mail 系统”。

例 3　某大学为进一步推进无纸化考试，欲开发一考试系统。系统管理员能够创建包括专业方向、课程编号、任课教师等相关考试基础信息，教师和学生进行考试相关的工作。系统与考试有关的主要功能如下。(2015 年 5 月试题一)

(1) 考试设置。教师制定试题(题目和答案)，制定考试说明、考试时间和提醒时间等考试信息，录入参加考试的学生信息，并分别进行存储。

(2) 显示并接收解答。根据教师设定的考试信息，在考试有效时间内向学生显示考试说明和题目，根据设定的考试提醒时间进行提醒，并接收学生的解答。

(3) 处理解答。根据答案对接收到的解答数据进行处理，然后将解答结果进行存储。

(4) 生成成绩报告。根据解答结果生成学生个人成绩报告，供学生查看。

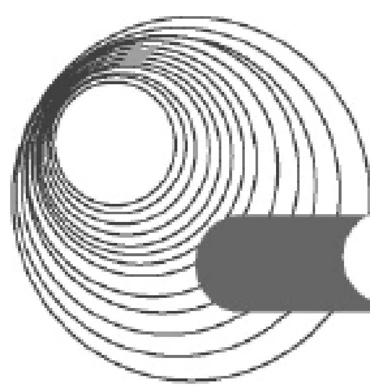
(5) 生成成绩单。对解答结果进行核算后生成课程成绩单供教师查看。

(6) 发送通知。根据成绩报告数据，创建通知数据并将通知发送给学生；根据成绩单数据，创建通知数据并将通知发送给教师。

现采用结构化方法对考试系统进行分析与设计，获得如图 1-6 所示的上下文数据流图和图 1-7 所示的 0 层数据流图。

【问题 1】(2 分)

使用说明中的词语，给出图 1-6 中的实体 E1～E2 的名称。



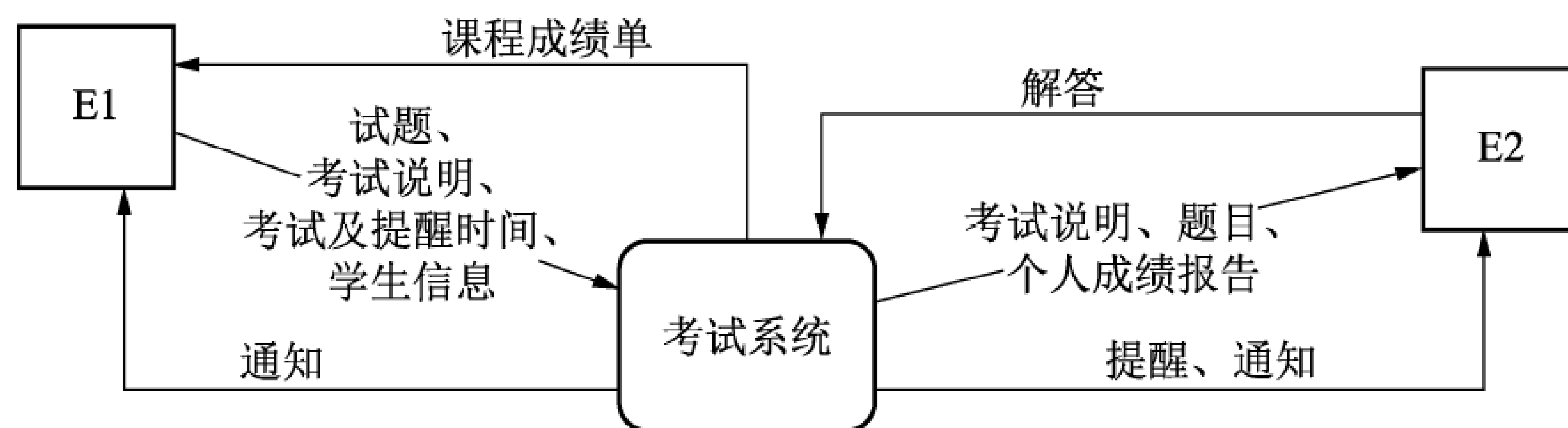


图 1-6　上下文数据流图

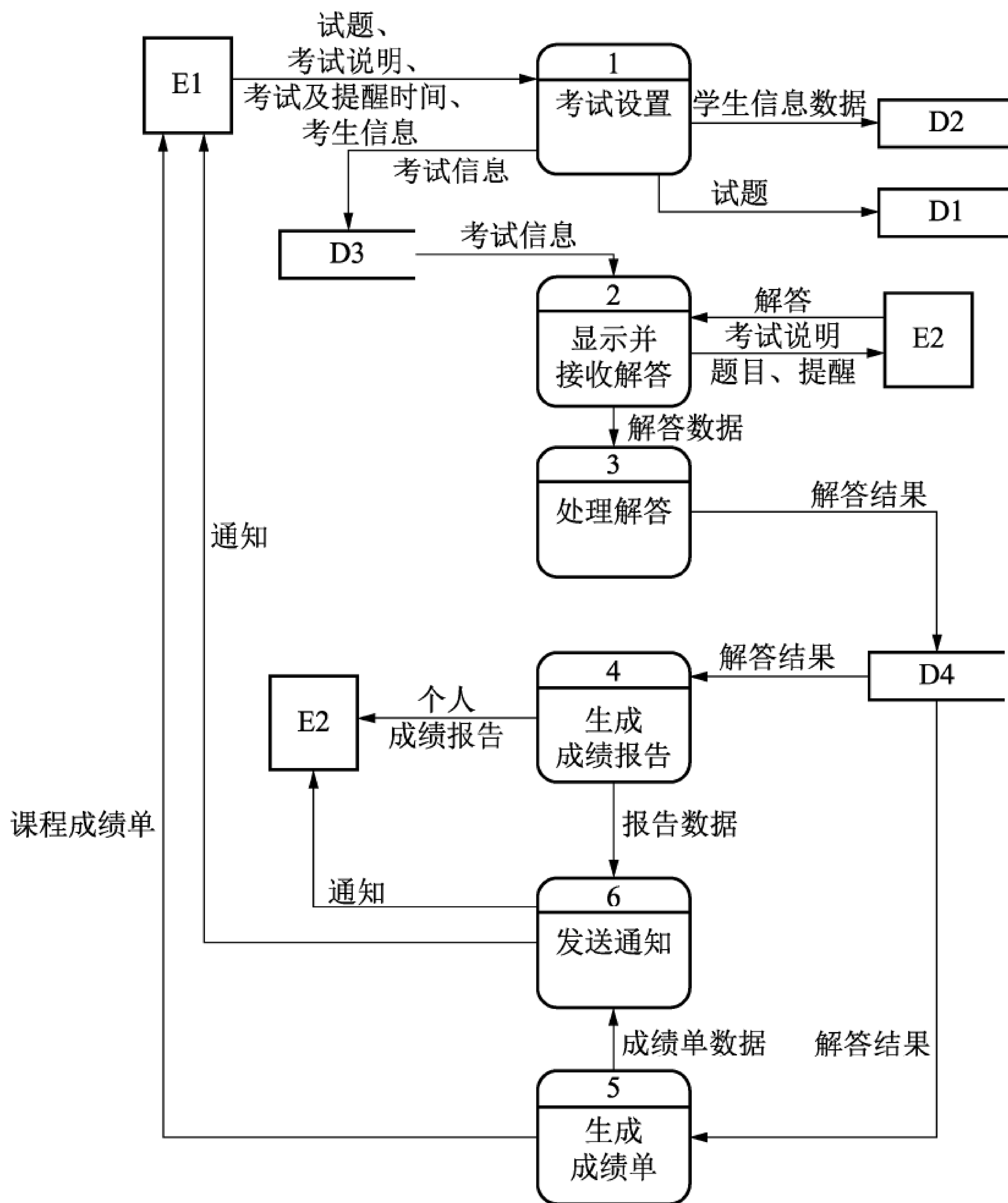


图 1-7　0 层数据流图

【问题 2】(4 分)

使用说明中的词语，给出图 1-7 中的数据存储 D1~D4 的名称。

【问题 3】(4 分)

根据说明和图中词语，补充图 1-7 中缺失的数据流及其起点和终点。

【问题 4】(5 分)

图 1-7 所示的数据流图中，功能(6)发送通知包含创建通知并发送给学生或教师。请分

解图 1-7 中的加工(6)，将分解出的加工和数据流填入答题纸的对应栏内。(注：数据流的起点和终点须使用加工的名称描述)

解析：

【问题 1】

根据题目说明，教师指定试题、考试说明、考试时间和考试题型，并录入学生信息，系统将考试说明、题目显示给学生，由学生解答。显然 E1 实体为“教师”，E2 实体为“学生”。

【问题 2】

教师将考试信息、试题、学生信息录入系统，根据数据流，显然 D1 为“试题”，D2 为“学生信息”，D3 为“考试信息”。学生解答后，由系统处理解答结果，并保存学生的解答结果，同时根据解答结果生成学生个人成绩报告，显然 D4 为“解答结果”。

【问题 3】

系统要向学生显示考试说明和题目，显然，缺少试题到加工 2 的数据流。学生做完题后，系统要根据答案对接收到的解答数据进行处理，这里缺少答案到处理 3 的数据流，根据题设知，答案包含在老师制定的试题中。

【问题 4】

根据题设，系统要根据成绩报告数据，创建通知数据并将通知发送给学生；根据成绩单数据，创建通知数据并将通知发送给教师。

答案：

【问题 1】

E1：教师。E2:学生。

【问题 2】

D1：试题。D2：学生信息。D3：考试信息。D4：解答结果。

【问题 3】

(1) 数据流名称：题目。起点：D1。终点：2(显示并接收解答)。

(2) 数据流名称：答案。起点：D1。终点：3(处理解答)。

【问题 4】

分解为：创建通知数据，发送通知给学生或老师。

数据流名称：通知数据。起点：创建通知数据。终点：发送通知给学生或老师。

例 4 某大型比萨加工和销售商为了有效管理生产和销售情况，欲开发一比萨信息系统。(2014 年 11 月试题一)

其主要功能如下。

(1) 销售。处理客户的订单信息，生成销售订单，并将其记录在销售订单表中。销售订单记录了订购者、所订购的比萨、期望的交付日期等信息。

(2) 生产控制。根据销售订单以及库存的比萨数量，制订比萨生产计划(包括生产哪些比萨、生产顺序和生产量等)，并将其保存在生产计划表中。

(3) 生产。根据生产计划和配方表中的比萨配方，向库存发出原材料申领单，将制作好的比萨的信息存入库存表中，以便及时进行交付。

(4) 采购。根据所需原材料及库存量，确定采购数量，向供应商发送采购订单，并将其记录在采购订单表中；得到供应商的供应量，将原材料数量记录在库存表中，在采购订单表中标记已完成采购的订单。

(5) 运送。根据销售订单将比萨交付给客户，并记录在交付记录表中。

(6) 财务管理。在比萨交付后，为客户开具费用清单，收款并出具收据；依据完成的采购订单给供应商支付原材料费用并出具支付细节；将收款和支付记录存入收支记录表中。

(7) 存储。检查库存的原材料、比萨和未完成订单，确定所需原材料。

现采用结构化方法对比萨信息系统进行分析与设计，获得如图 1-8 所示的上下文数据流图和图 1-9 所示的 0 层数据流图。

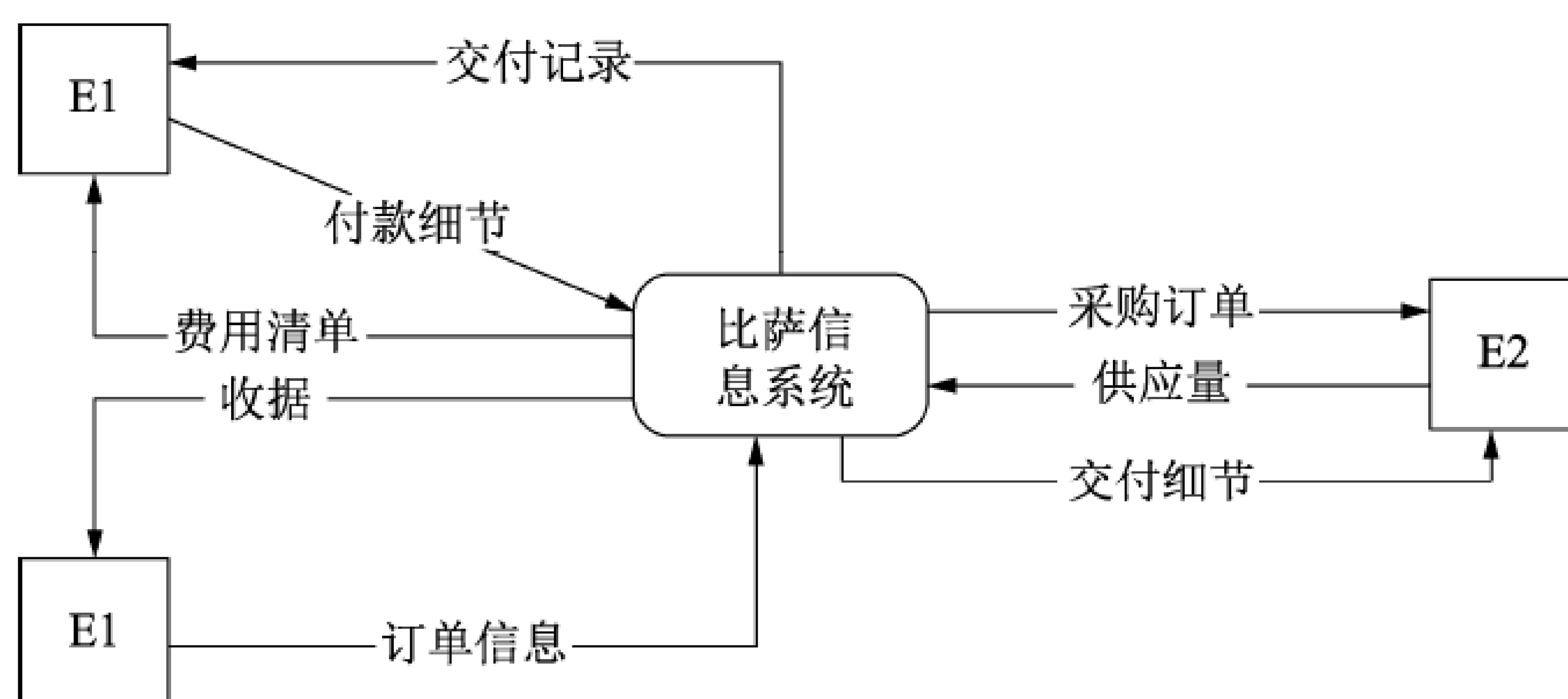


图 1-8　上下文数据流图

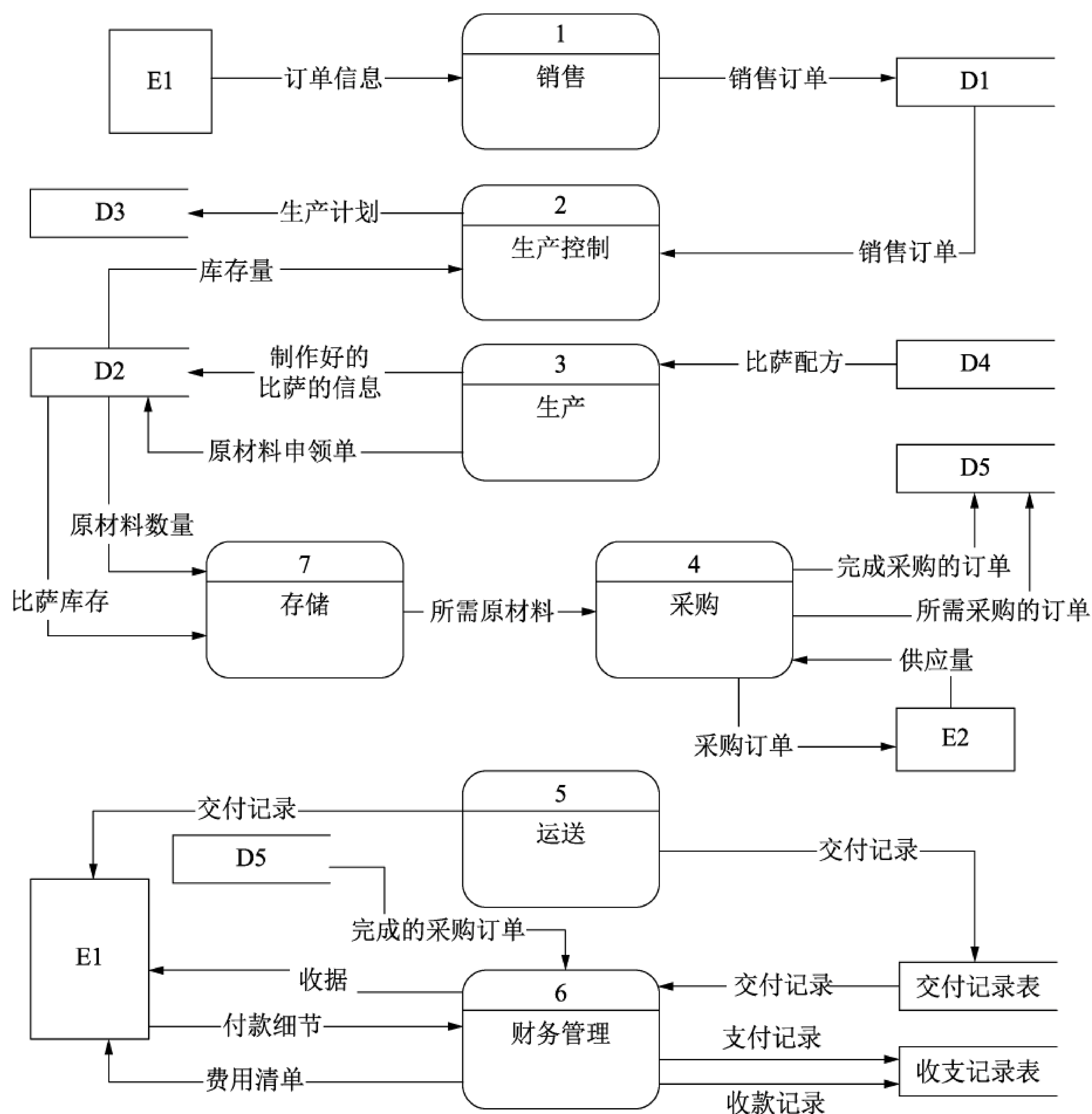


图 1-9　0 层数据流图

【问题 1】(4 分)

根据说明中的词语，给出图 1-8 中的实体 E1~E2 的名称。

【问题 2】(5 分)

根据说明中的词语，给出图 1-9 中的数据存储 D1~D5 的名称。

【问题 3】(6 分)

根据说明中的词语，补充图 1-9 中缺失的数据流及其起点和终点。

解析：

该题以比萨信息系统为载体来考查学生对数据流图知识点的掌握程度。从题目的问答形式上来看，和往年相似，要求补充外部实体、补充缺失数据流、找出外部存储。

【问题 1】

根据 0 层数据流中财务管理为客户开具费用清单数据流可知，E1 实体为客户；从向供应商发送采购订单、得到供应商的供应量可知，E2 实体为供应商。

【问题 2】

根据题中对功能的描述，以及 0 层数据流：生产控制，制订生产计划，并将其保存在生产计划表中可知，D3 为生产计划表；生产，将制作好的比萨的信息存入库存表中，可知 D2 为库存表；采购，在采购订单表中标记已完成的订单，可知 D5 为采购订单表；生产，由“根据生产计划和配方表中的比萨配方，向库存发出原料申请”，可知 D4 为配方表；根据“处理客户的订单信息，生成销售订单，并将其记录在销售订单表中”，可知 D1 为销售订单表。

【问题 3】

本题考查补充分层 DFD 中的数据流。在分层 DFD 中，需要保持父图和子图的平衡，即父图中某加工的输入输出数据流必须与其子图的输入输出数据流在数量和名字上相同，或者父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流，而子图中组成这些数据流的数据项全体正好是父图中的一个数据流。

由“财务管理”段中“依据完成的采购订单给供应商支付原材料费用并出具支付细节”的描述，存在一个起点为财务管理，终点为供应商即 E2 的数据流，即支付细节数据流；由“运送”段中“根据销售订单将比萨交付给客户，并记录在交付记录表中”，可知存在一个由“销售订单”指向“运送”的数据流，即销售订单数据流；由“采购”段中“根据所需原材料及库存量，确定采购数量”，可知存在由“库存表”指向“采购”的数据流，数据流为库存量；根据“存储”段中“检查库存的原材料、比萨和未完成订单，确定所需原料”可知，存在一个由“订单表”指向“存储”的数据流，数据流为未完成的订单。

答案：

【问题 1】

E1：客户。E2：供应商。

【问题 2】

D1：销售订单表。D2：库存表。D3：生产计划表。D4：配方表。D5：采购订单表。

【问颗 3】

(1) 数据流名称：支付细节。起点：财务管理。终点：E2。

(2) 数据流名称：销售订单。起点：销售订单表。终点：5(运送)。

(3) 数据流名称：生产计划。起点：D3。终点：3(生产)。

(4) 数据流名称：库存量。起点：D2。终点：4(采购)。

(5) 数据流名称：原材料数量。起点：4 采购。终点：D2。

(6) 数据流名称：未完成订单。起点：销售订单表。终点：7(存储)。

例 5 某巴士维修连锁公司欲开发巴士维修系统，以维护与维修相关的信息，该系统的主要功能如下。(2014 年 5 月试题一)

(1) 记录巴士 ID 和维修问题。巴士到车库进行维修，系统将巴士基本信息和 ID 记录在巴士列表文件中，将待维修机械问题记录在维修记录文件中，并生成维修订单。

(2) 确定所需部件。根据维修订单确定维修所需部件，并在部件清单中进行标记。

(3) 完成维修。机械师根据维修记录文件中的待维修机械问题，完成对巴士的维修，登记维修情况；将机械问题维修情况记录在维修记录文件中，将所用部件记录在部件清单中，并将所用部件清单发送给库存管理系统以对部件使用情况进行监控，巴士司机可查看已维修机械问题。

(4) 记录维修工时。将机械师提供的维修工时记录在人事档案中，将维修总结发送给主管进行绩效考核。

(5) 计算维修总成本。计算部件清单中实际所用部件、人事档案中所用维修工时的总成本；将维修工时和所用部件成本详细信息给会计进行计费。

现采用结构化方法对巴士维修系统进行分析与设计，获得如图 1-10 所示的上下文数据流图和图 1-11 所示的 0 层数据流图。

【问题 1】(5 分)

使用说明中的词语，给出图 1-10 中的实体 E1~E5 的名称。

【问题 2】(4 分)

使用说明中的词语，给出图 1-11 中的数据存储 D1~D4 的名称。

【问题 3】(3 分)

说明图 1-11 中所存在的问题。

【问题 4】(3 分)

根据说明和图中术语，采用补充数据流的方式，改正图 1-11 中的问题。要求给出所补充数据流的名称、起点和终点。

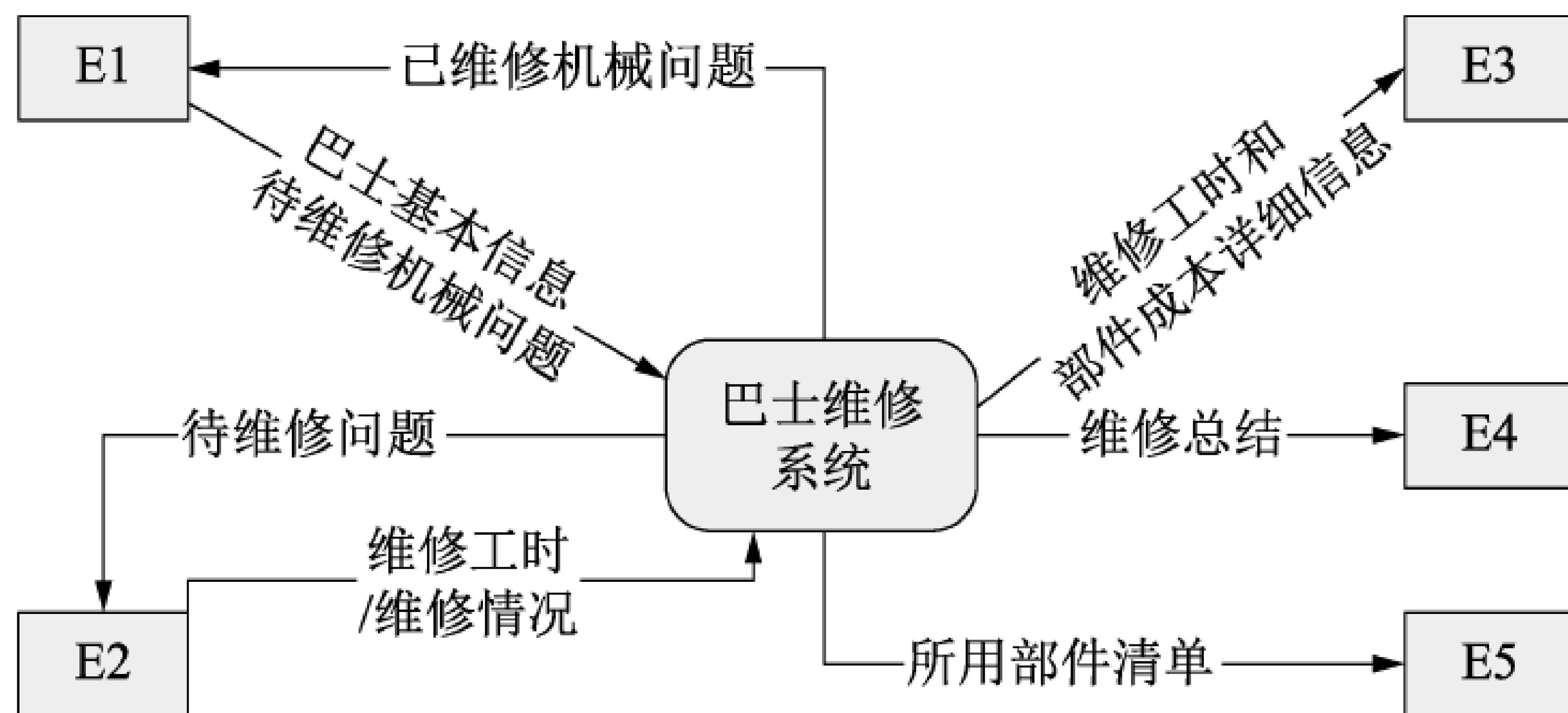


图 1-10 上下文数据流图

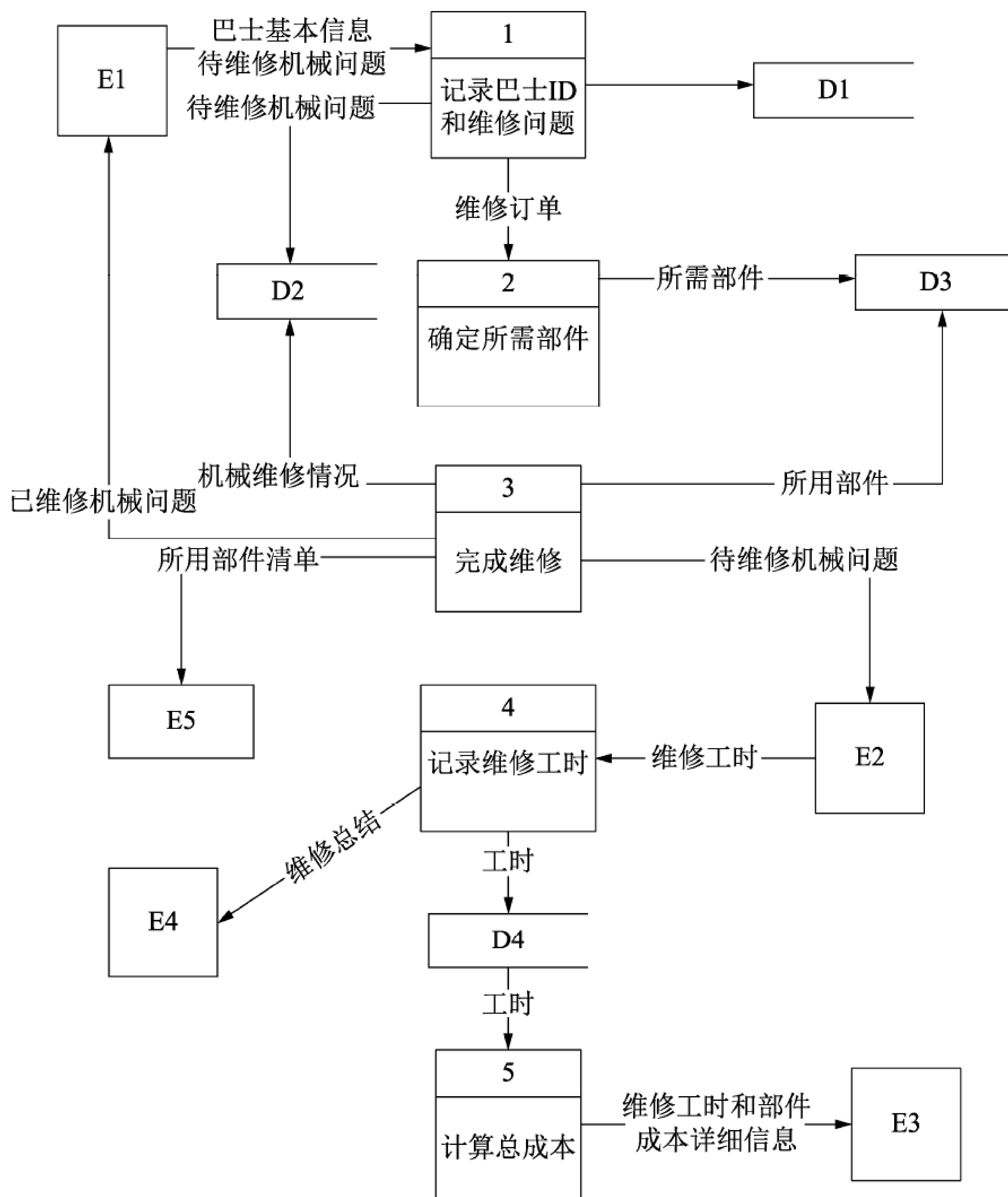


图 1-11　0 层数据流图

解析：

本题考查数据流图(DFD)的应用，采用结构化方法进行系统分析与设计，是一道传统题目，要求考生细心分析题目中所描述的内容。DFD 是一种便于用户理解、分析系统数据流程的图形化建模工具，是系统逻辑模型的重要组成部分。

【问题 1】

本问题考查顶层 DFD。顶层 DFD 一般用来确定系统边界。结合图 1-10 上下文数据流图以及根据系统功能的描述(3)“巴士司机可查看已维修问题”可知，E1 为“巴士司机”；根据系统功能描述(4)“机械师提供的维修工时记录在人事档案中”和“维修总结发送给主管进行考核”可知，E2 为“机械师”，E4 为“主管”；根据系统功能描述(5)“维修工时和所用成本详细信息给会计进行计费”可知，E3 为“会计师”；由“所用部件清单发送给库存管理系统”可推断，E5 为“库存管理系统”。

【问题 2】

本问题考查 DFD 中数据存储的确定。本题涉及的数据存储有 4 个，即巴士列表文件、维修记录文件、部件清单和人事档案。接下来需要结合图 1-11 所示的 0 层数据流进行确定。D1 与“记录巴士 ID 和维修问题”有关，根据题意，D1 是“巴士列表文件”；待维修机械问题记录在 D2 中，可知，D2 为“维修记录文件”；所需部件记录在 D3 中，由题意可知 D3 为“部件清单”；D4 记录的是工时，由题意可知 D4 是“人事档案”。

【问题 3】

本问题考查 DFD 中的数据流。在 DFD 中，需要保持父图和子图的平衡，即父图中某加工的输入/输出数据流必须与其子图的输入/输出数据流在数量和名字上相同，或者父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流，而子图中组成这些数据流的数据项全体正好是父图中的一个数据流。由功能(5)的描述“计算部件清单中实际所用部件、人事档案中所用维修工时的总成本”可知，缺少关于实际所用部件数据流，因此增加一个数据流“实际所用部件”，起点为 D3，终点为 5(计算总成本)。根据“完成维修”功能中的描述，机械师可根据维修记录文件中的待维修机械问题，完成巴士维修，因此缺少数据流“待维修机械问题”，起点为 D2，终点为 3(完成维修)。

【问题 4】

根据问题 3 解析，补充以下数据流：实际所用部件，起点：D3，终点：5(计算总成本)；待维修机械问题，起点：D2，终点：3(完成维修)。

答案：

【问题 1】

E1：巴士司机。

E2：机械师。

E3：会计师。

E4：主管。

E5：库存管理系统。

【问题 2】

D1：巴士列表文件。

D2：维修记录文件。

D3：部件清单。

D4：人事档案。

【问题 3】

缺失以下数据流。

(1) 数据流名称：待维修机械问题。起点：D2。终点：3(完成维修)。

(2) 数据流名称：实际所用部件。起点：D3。终点：5(计算总成本)。

【问题 4】

补充以下数据流。

(1) 数据流名称：待维修机械问题。起点：D2。终点：3(完成维修)。

(2) 数据流名称：实际所用部件。起点：D3。终点：5(计算总成本)。

例 6 某大学欲开发一个基于 Web 的课程注册系统，该系统的主要功能如下。(2013 年

11 月试题一)

1) 验证输入信息

(1) 检查学生信息：检查学生输入的所有注册所需信息。如果信息不合法，返回学生信息不合法提示；如果合法，输出合法学生信息。

(2) 检查学位考试结果：检查学生提供的学位考试结果。如果不合法，返回学位考试结果不合法提示；如果合法，检查该学生注册资格。

(3) 检查学生注册资格：根据合法学生信息和合法学位考试结果，检查该学生对欲选课程的注册资格。如果无资格，返回无注册资格提示；如果有注册资格，则输出注册学生信息(包含选课学生标识)和欲注册课程信息。

2) 处理注册申请

(1) 存储注册信息：将注册学生信息记录在学生库。

(2) 存储所注册课程：将选课学生标识与欲注册课程进行关联，然后存入课程库。

(3) 发送注册通知：从学生库中读取注册学生信息，从课程库中读取所注册课程信息，给学生发送接收提示；给教务人员发送所注册课程信息和已注册学生信息。

现采用结构化方法对课程注册系统进行分析与设计，获得如图 1-12 所示的 0 层数据流图和图 1-13 所示的 1 层数据流图。

【问题 1】(2 分)

使用说明中的词语，给出图 1-12 中的实体 E1 和 E2 的名称。

【问题 2】(2 分)

使用说明中的词语，给出图 1-13 中的数据存储 D1 和 D2 的名称。

【问题 3】(8 分)

根据说明和图中术语。补充图 1-13 中缺失的数据流及其起点和终点。

【问题 4】(3 分)

根据补充完整的图 1-12 和图 1-13，说明上层的哪些数据流是由下层的哪些数据流组合而成。

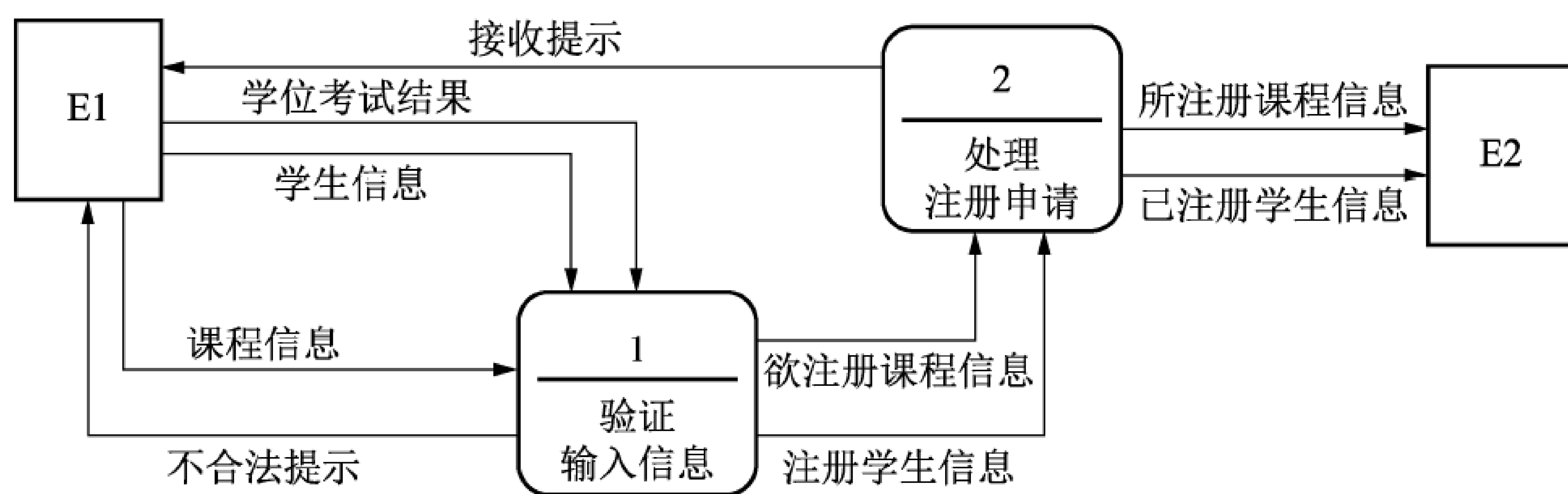
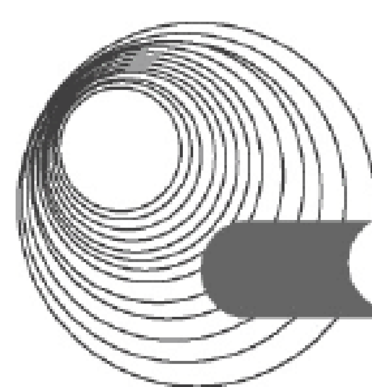


图 1-12　某课程注册系统 0 层数据流图



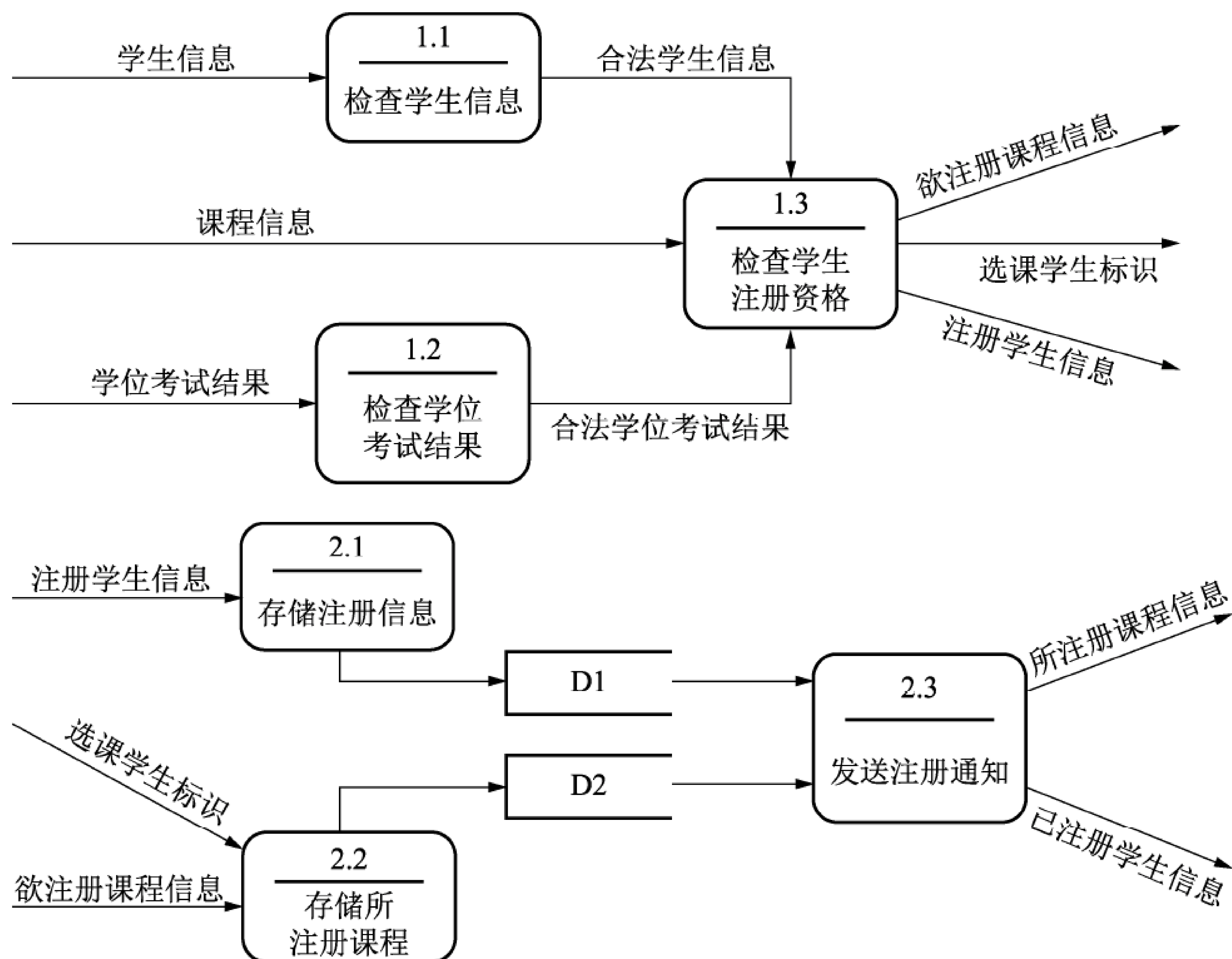


图 1-13　某课程注册系统 1 层数据流图

解析：

该题以 Web 注册系统为载体来考核考生对数据流图知识点的掌握程度。从题目的问答形式上来看，和往年相似，要求补充外部实体、补充缺失数据流、找出外部存储。解答这类问题，有以下两个原则。

(1) 紧扣试题系统说明部分，数据流图与系统说明有着严格的对应关系，系统说明部分的每一句话都能对应到图中来，解题时可以一句一句地对照图来分析。

(2) 数据的平衡原则，这一点在解题过程中也是至关重要的。数据平衡原则有两方面的含义，一方面是分层数据流图父子图之间的数据流平衡原则，另一方面是每张数据流图中输入与输出数据流的平衡原则。

【问题 1】

根据 0 层数据流图的课程注册系统可知：根据说明中向验证输入信息发送学位考试结果、学生信息、课程信息等可知 E1 为学生。同样，根据说明，经过处理注册申请向 E2 发送注册课程信息和已注册信息，可知 E2 为教务人员。

【问题 2】

根据题目中说明，存储注册信息时，“将注册学生信息记录在学生库”，可知 D1 为学生库；存储注册课程时，关联后“存入课程库”，可知 D2 为课程库。

【问题 3】

对应图 1-12 中处理 1(验证输入信息)的输出数据流“不合法提示”，不难发现，在图 1-13 中，处理 1.1 缺少了到实体学生的输出数据流“学生信息不合法提示”；处理 1.2 缺

少了到实体学生的输出数据流“无注册资格提示”；处理 1.3 缺少了到实体学生的输出数据流“学位考试结果不合法提示”。再查图 1-12 中处理 2，其输出数据流有三条，而图 1-13 中对图 1-12 处理 2 中，只包含了“所注册课程信息”和“已注册学生信息”两条数据流，缺失了“接收提示”。

【问题 4】

本问题考查数据流的分解与组合。图 1-13 中对于说明中的功能出现了“学生信息不合法提示”“无注册资格提示”和“学位考试结果不合法提示”三条数据流，说明图 1-12 中的数据流“不合法提示”是由这三条数据流组合而成。

图 1-12 中注册学生信息在图 1-13 中进一步分出注册学生信息和选课学生标识，即图 1-12 中注册学生信息是注册学生信息和选课学生标识的并集。

答案：

【问题 1】

E1：学生。E2：教务人员。

【问题 2】

D1：学生库。D2：课程库。

【问题 3】

图 1-13 中缺失的数据流及其起点和终点如表 1-1 所示。

表 1-1　例 6 问题 3 表

数 据 流	起　点	终　点
学生信息不合法提示	1.1 或 检查学生信息	E1 或 学生
无注册资格提示	1.3 或 检查学生注册资格	E1 或 学生
学位考试结果不合法提示	1.2 或 检查学位考试结果	E1 或 学生
接收提示	2.3 或 发送注册通知	E1 或 学生

【问题 4】

图 1-12 中不合法提示分解为图 1-13 中的三条数据流的组合：学生信息不合法提示、无注册资格提示、学位考试结果不合法提示。

图 1-12 中注册学生信息对应图 1-13 中注册学生信息和选课学生标识。

例 7　阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2013 年 5 月试题一)

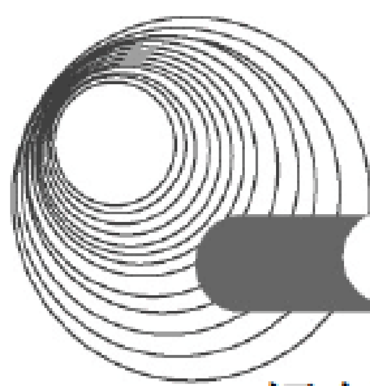
【说明】

某慈善机构欲开发一个募捐系统，以跟踪、记录为事业或项目向目标群体进行募捐而组织的集体性活动。该系统的主要功能如下所述。

(1) 管理志愿者。根据募捐任务给志愿者发送加入邀请、邀请跟进、工作任务；管理志愿者提供的邀请响应、志愿者信息、工作时长、工作结果等。

(2) 确定募捐需求和收集所募捐赠(资金及物品)。根据需求提出募捐任务、活动请求和捐赠请求，获取所募集的资金和物品。

(3) 组织募捐活动。根据活动请求，确定活动时间范围。根据活动时间，搜索场馆，向



场馆发送场馆可用性请求，获得场馆可用性。根据活动时间和地点推广募捐活动，根据相应的活动信息举办活动，从募捐机构获取资金并向其发放赠品。获取和处理捐赠，根据捐赠请求，提供所募集的捐赠；处理与捐赠人之间的交互，即录入捐赠人信息，处理后存入捐赠人信息表；从捐赠人信息表中查询捐赠人信息，向捐赠人发送募捐请求，并将已联系的捐赠人存入已联系的捐赠人表。根据捐赠请求进行募集，募得捐赠后，将捐赠记录存入捐赠表；对捐赠记录进行处理后，存入已处理捐赠表，向捐赠人发送致谢函，根据已联系的捐赠人和捐赠记录进行跟踪，将捐赠跟进情况发送给捐赠人。

先采用结构化方法对募捐系统进行分析与设计，获得如图 1-14~图 1-16 所示的分层数据流图。

【问题 1】(4 分)

使用说明中的词语，给出图 1-14 中的实体 E1~E4 的名称。

【问题 2】(7 分)

在建模 DFD 时，需要对有些复杂加工(处理)进行进一步精化，图 1-15 为图 1-14 中处理 3 的进一步细化的 1 层数据流图，图 1-16 为图 1-15 中 3.1 的进一步细化的 2 层数据流图。补全图 1-14 中加工 P1~P3 的名称和图 1-15 与图 1-16 中缺少的数据流。

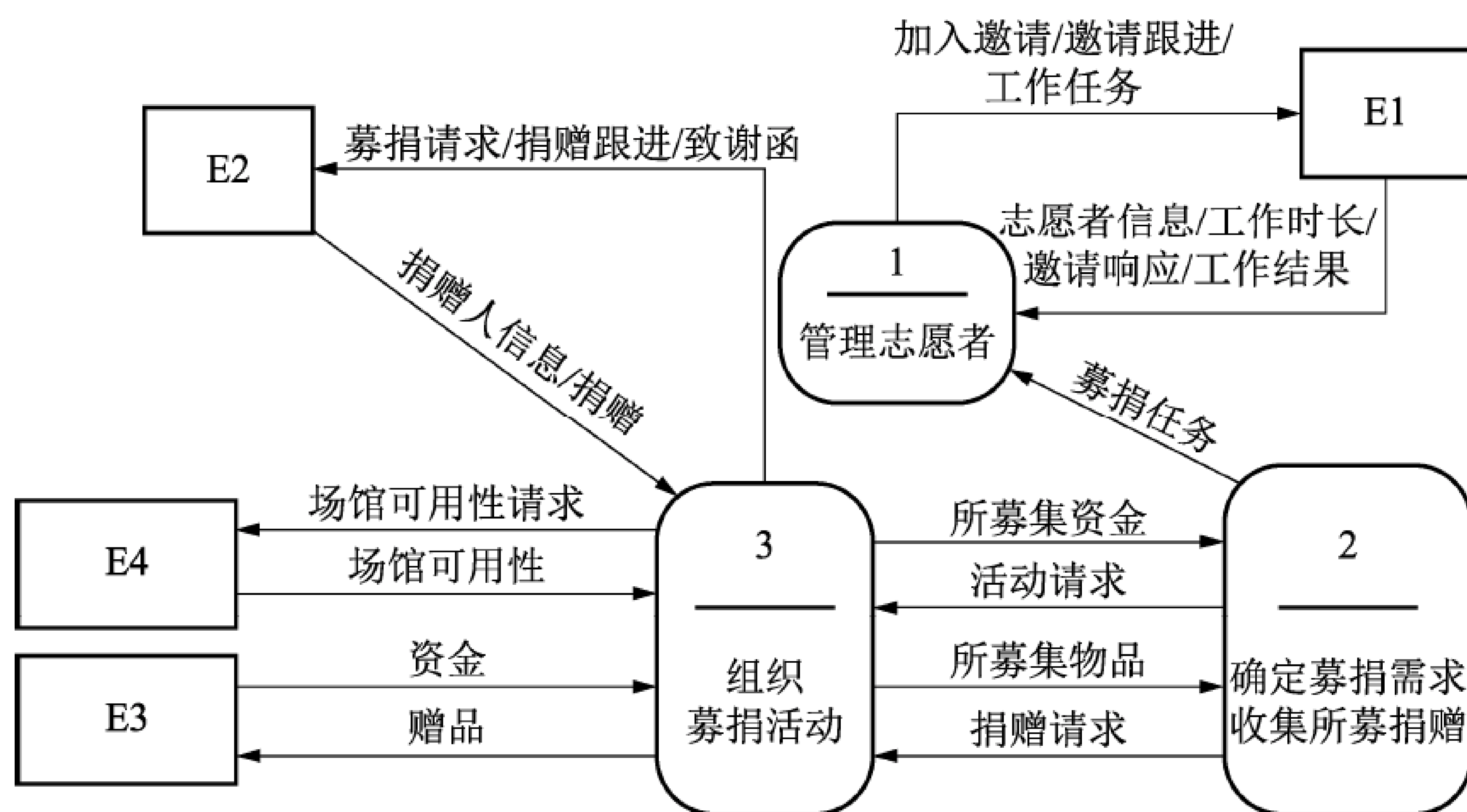


图 1-14　0 层数据流图

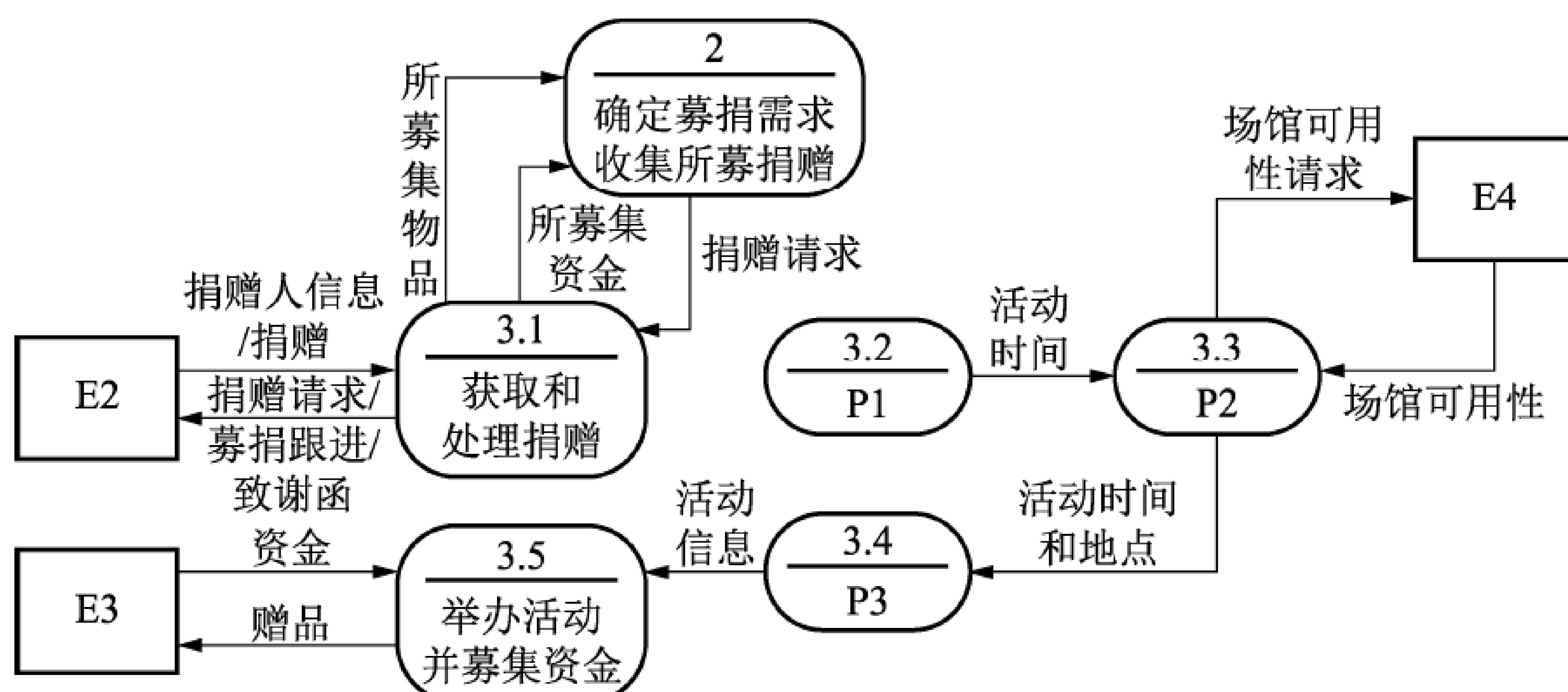


图 1-15　1 层数据流图

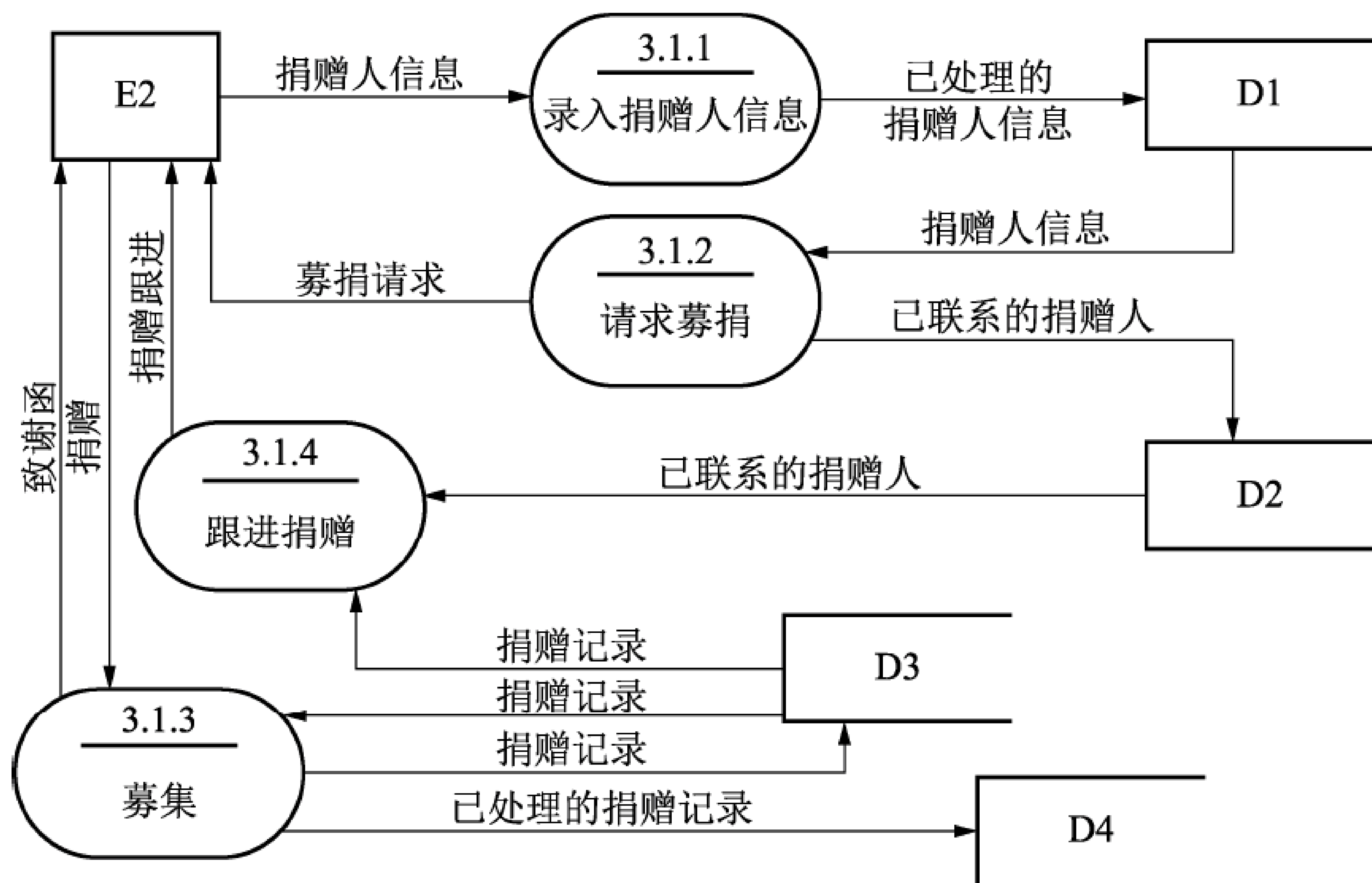


图 1-16　2 层数据流图

【问题 3】(4 分)

使用说明中的词语，给出图 1-16 中的数据存储 D1~D4 的名称。

解析：

该题以募捐系统为载体来考核考生对数据流图知识点的掌握程度。从题目的问答形式上来看，和往年相似，要求补充外部实体、补充缺失数据流、找出外部存储。解答这类问题的原则参考例 6。

【问题 1】

根据募捐系统“根据募捐任务给志愿者发送加入邀请、邀请跟进、工作任务”和“管理志愿者提供的邀请响应、志愿者信息、工作时长、工作结果等”可知，E1 为志愿者。根据募捐系统录入捐赠人信息、向捐赠人发送募捐请求、向捐赠人发送致谢函等可知，E2 为捐赠人。根据募捐系统“从募捐机构获取资金并向其发放赠品”可知，E3 为募捐机构。根据募捐系统“场馆发送可用性请求，获得场馆可用性”可知，E4 为场馆。

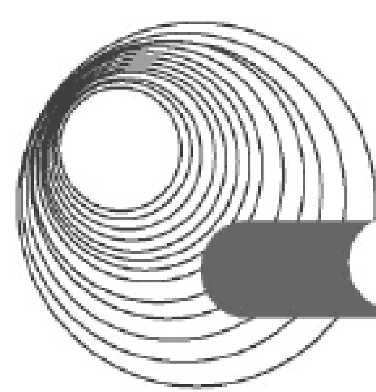
【问题 2】

根据 1 层数据流图中 P1 的输出流“活动时间”再结合说明可知，P1 为“确定活动时间范围”。从 P2 的输入流“活动时间”和输出流“场馆可用性请求”和“活动时间和地点”可知，P2 为“搜索场馆”。从说明中“根据活动时间和地点推广募捐活动，根据相应的活动信息举办活动”，再结合 P3 的输入输出流可知，P3 为“推广募捐活动”。

比较 0 层数据流图和 1 层数据流图中的数据流可知，P1 只有输出流，故缺少输入流，根据说明可知需要根据活动请求才能确定 P1，故该数据流为“活动请求”。在 0 层数据流图中，活动请求的起始加工为“确定募捐需求、收集所募捐赠”。根据说明中的“根据捐赠请求进行募集”和 2 层数据流图可知，募集加工需要的来自 E2 的数据流不仅仅是捐赠，还有捐赠请求，故可知 2 层数据流图中缺少的数据流。

【问题 3】

根据最后的说明和 2 层数据流图可知，D1 为捐赠人信息表，D2 为已联系的捐赠人表，



D3 为捐赠表，D4 为已处理捐赠表。

答案：

【问题 1】

E1：志愿者。E2：捐赠人。E3：募捐机构。E4：场馆。

【问题 2】

P1：确定活动时间范围。P2：搜索场馆。P3：推广募捐活动。

图 1-14 中缺少的数据流如下。

名称：活动请求。起点：确定募捐需求、收集所募捐赠。终点：P1。

【问题 3】

D1：捐赠人信息表。D2：已联系的捐赠人表。D3：捐赠表。D4：已处理捐赠表。

1.1.3 同步练习

1. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2012 年 11 月试题一)

【说明】

某电子商务系统采用以数据库为中心的集成方式改进购物车的功能，详细需求如下。

(1) 加入购物车。顾客浏览商品，点击加入购物车，根据商品标识从商品表中读取商品信息，并更新购物车表。

(2) 浏览购物车。顾客提交浏览购物车请求后，显示出购物车表中的商品信息。

(3) 提交订单。顾客点击提交订单请求，后台计算购物车表中商品的总价(包括运费)加入订单表，将购物车表中的商品状态改为待付款，显示订单详情。若商家改变价格，则刷新后可看到更改后的价格。

(4) 改变价格。商家查看订购自家商品的订单信息，根据特殊优惠条件修改价格，更新订单表中的商品价格。

(5) 付款。顾客点击付款后，系统先根据顾客表中关联的支付账户，将转账请求(验证码、价格等)提交给支付系统(如信用卡系统)进行转账；然后根据转账结果返回支付状态并更改购物车表中商品的状态。

(6) 物流跟踪。商家发货后，需按订单标识添加物流标识(物流公司、运单号)；然后可根据顾客或商家的标识以及订单标识，查询订单表中的物流标识，并从相应物流系统查询物流信息。

(7) 生成报表。根据管理员和商家设置的报表选项，从订单表、商品表以及商品分类表中读取数据，调用第三方服务 Crystal Reports 生成相关报表。

(8) 维护信息。管理员维护(增、删、改、查)顾客表、商品分类表和商品表中的信息。

现采用结构化方法实现上述需求，在系统分析阶段得到如图 1-17 所示的顶层数据流图和如图 1-18 所示的 0 层数据流图。

【问题 1】(4 分)

使用说明中的词语，给出图 1-17 中的外部实体 E1~E4 的名称。



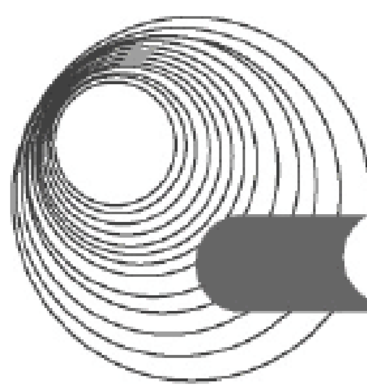
图 1-17　顶层数据流图



图 1-18　0 层数据流图

【问题 2】(4 分)

使用说明中的词语，给出图 1-18 中的数据存储 D1~D4 的名称。



【问题 3】(4 分)

图 1-18 中缺失了数据流，请用说明或图 1-18 中的词语，给出其起点和终点。

【问题 4】(3 分)

根据说明，给出数据流“转账请求”“顾客订单物流查询请求”和“商家订单物流查询请求”的各组成数据项。

2. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2012 年 5 月试题一)

【说明】

某学校开发图书管理系统，以记录图书馆藏书及其借出和归还情况，提供给借阅者借阅图书功能，提供给图书馆管理员管理和定期更新图书表功能。主要功能的具体描述如下。

(1) 处理借阅。借阅者要借阅图书时，系统必须对其身份(借阅者 ID)进行检查。通过与教务处维护的学生数据库、人事处维护的职工数据库中的数据进行比对，以验证借阅者 ID 是否合法。若合法，则检查借阅者在逾期未还图书表中是否有逾期未还图书，以及罚金表中的罚金是否超过限额。如果没有逾期未还图书并且罚金未超过限额，则允许借阅图书，更新图书表，并将借阅的图书存入借出图书表。借阅者归还所借图书时，先由图书馆管理员检查图书是否缺失或损坏，若是，则对借阅者处以相应罚金并存入罚金表；然后，检查所还图书是否逾期，若是，执行“处理逾期”操作；最后，更新图书表，删除借出图书表中的相应记录。

(2) 维护图书。图书馆管理员查询图书信息；在新进图书时录入图书信息，存入图书表；在图书丢失或损坏严重时，从图书表中删除该图书记录。

(3) 处理逾期。系统在每周一统计逾期未还图书，逾期未还的图书按规则计算罚金，并记入罚金表，并给有逾期未还图书的借阅者发送提醒消息。借阅者在借阅和归还图书时，若罚金超过限额，管理员收取罚金，并更新罚金表中的罚金额度。

现采用结构化方法对该图书管理系统进行分析与设计，获得如图 1-19 所示的顶层数据流图和如图 1-20 所示的 0 层数据流图。

【问题 1】(4 分)

使用说明中的词语，给出图 1-19 中的外部实体 E1~E4 的名称。

【问题 2】(4 分)

使用说明中的词语，给出图 1-20 中的数据存储 D1~D4 的名称。

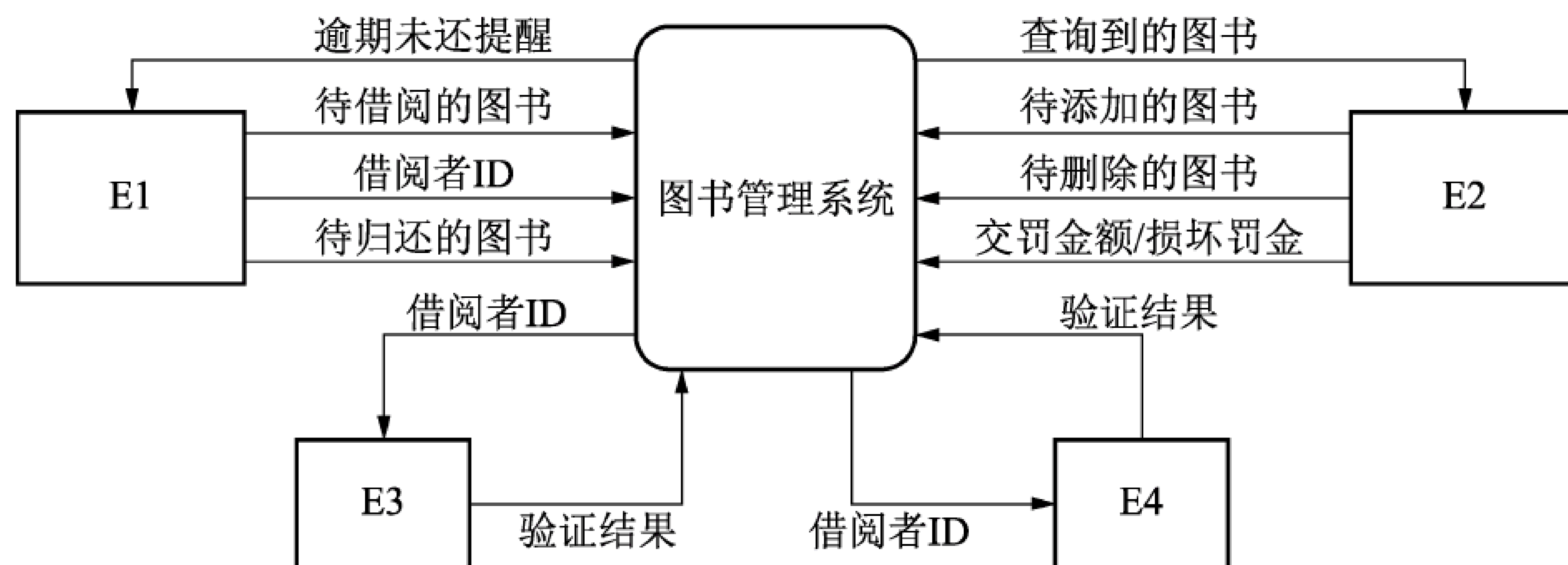


图 1-19 顶层数据流图

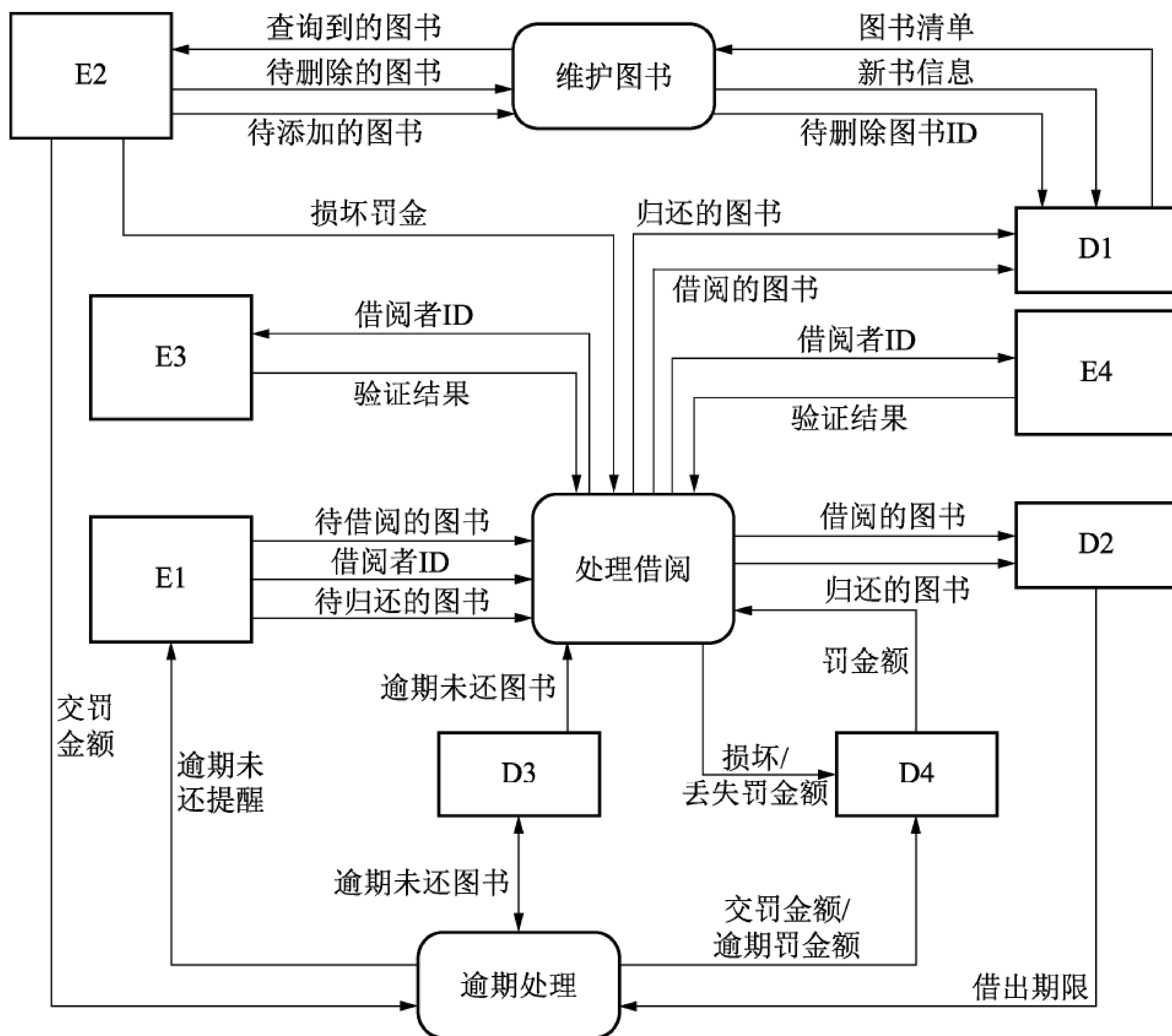


图 1-20　0 层数据流图

【问题 3】 (5 分)

在 DFD 建模时，需要对某些复杂加工(处理)进行进一步精化，绘制下层数据流图。针对图 1-20 中的加工“处理借阅”，在 1 层数据流图中应分解为哪些加工？(使用说明中的术语)

【问题 4】(2 分)

说明问题 3 中绘制 1 层数据流图时要注意的问题。

3. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2011 年 11 月试题一)

【说明】

某公司欲开发招聘系统以提高招聘效率，其主要功能如下。

(1) 接受申请。验证应聘者所提供的自身信息是否完整、是否说明了应聘职位，受理验证合格的申请，给应聘者发送致谢信息。

(2) 评估应聘者。根据部门经理设计的职位要求，审查已经受理的申请；对未被录用的应聘者进行谢绝处理，将未被录用的应聘者信息存入未录用的应聘者表，并给其发送谢绝决策；对录用的应聘者进行职位安排评价，将评价结果存入评价结果表，并给其发送录用决策，发送录用职位和录用者信息给工资系统。

现采用结构化方法对招聘系统进行分析和设计，获得如图 1-21 所示的顶层数据流图、如图 1-22 所示的 0 层数据流图和如图 1-23 所示的 1 层数据流图。

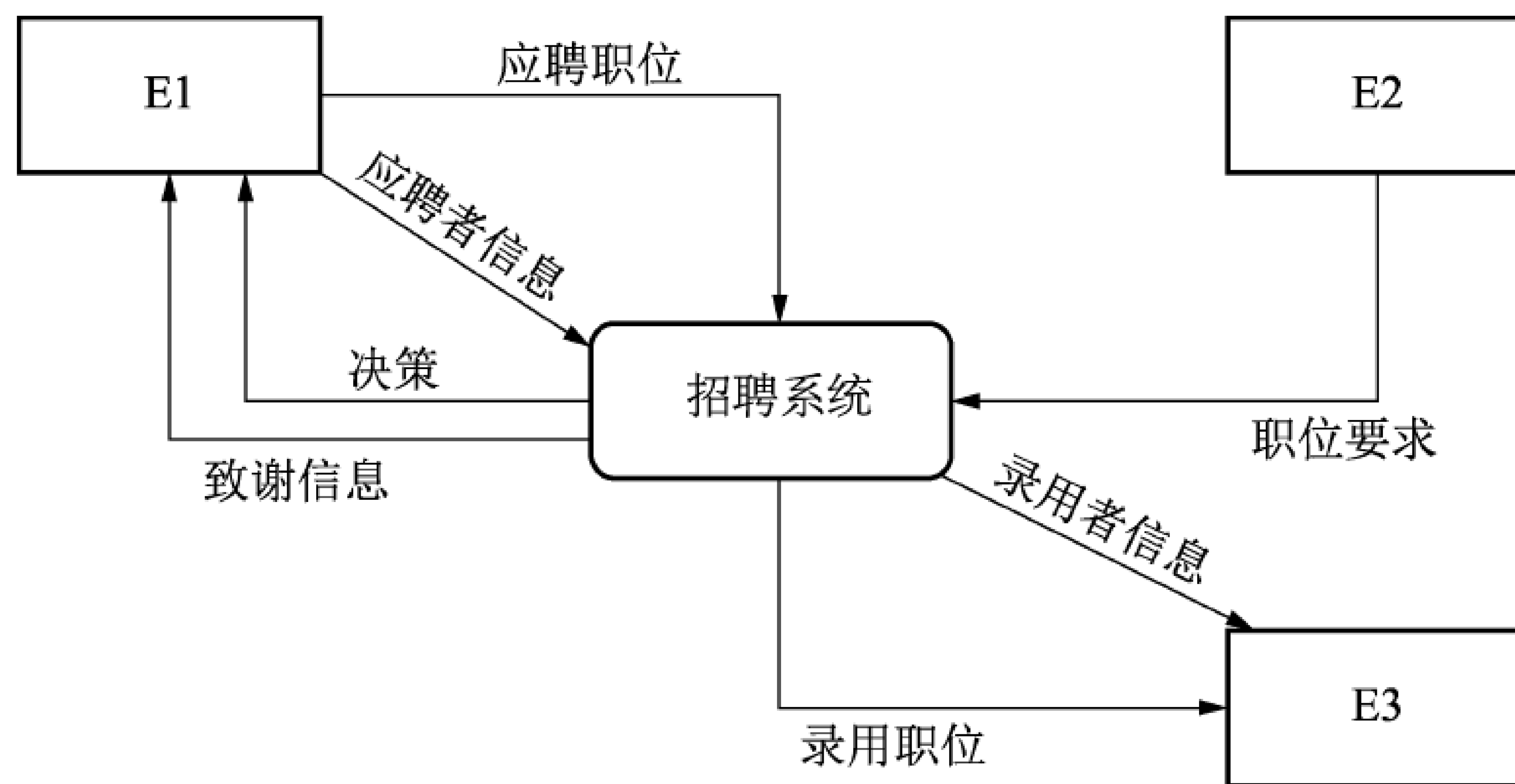


图 1-21　顶层数据流图

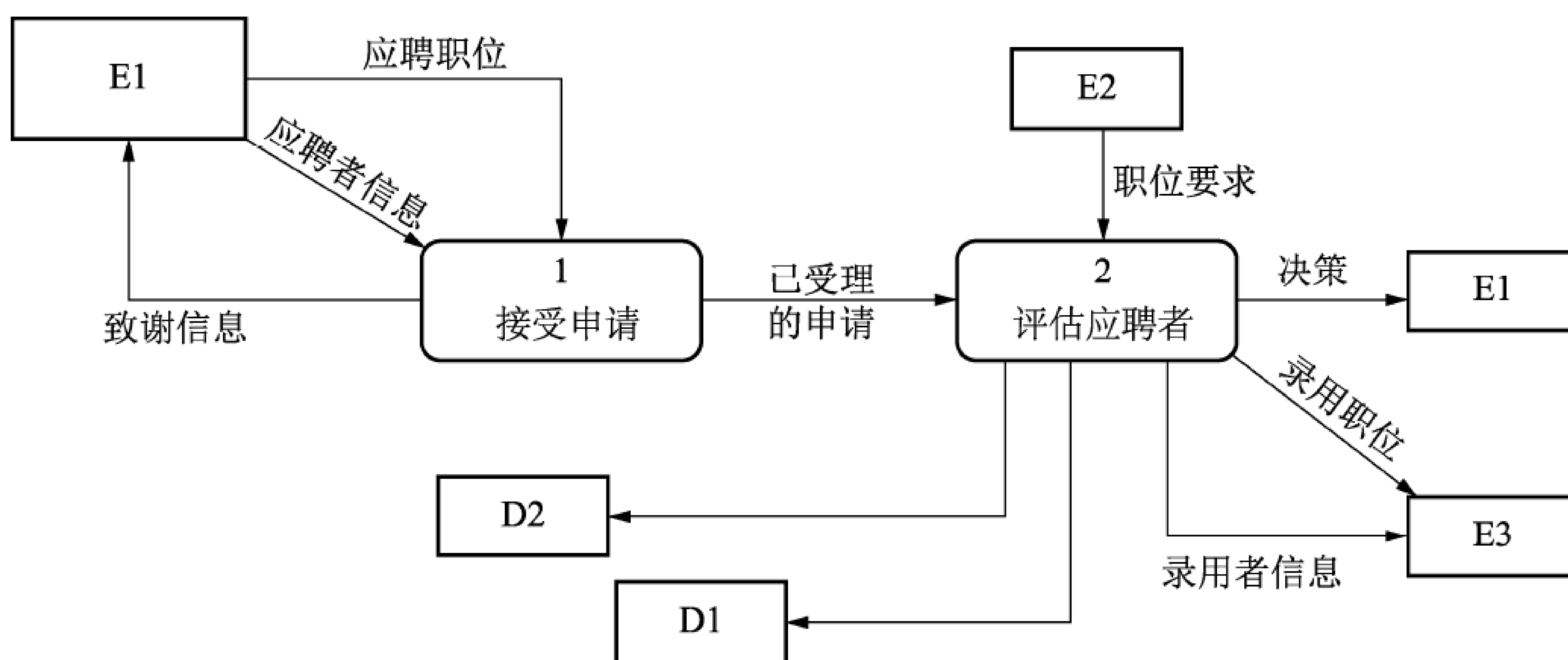


图 1-22　0 层数据流图

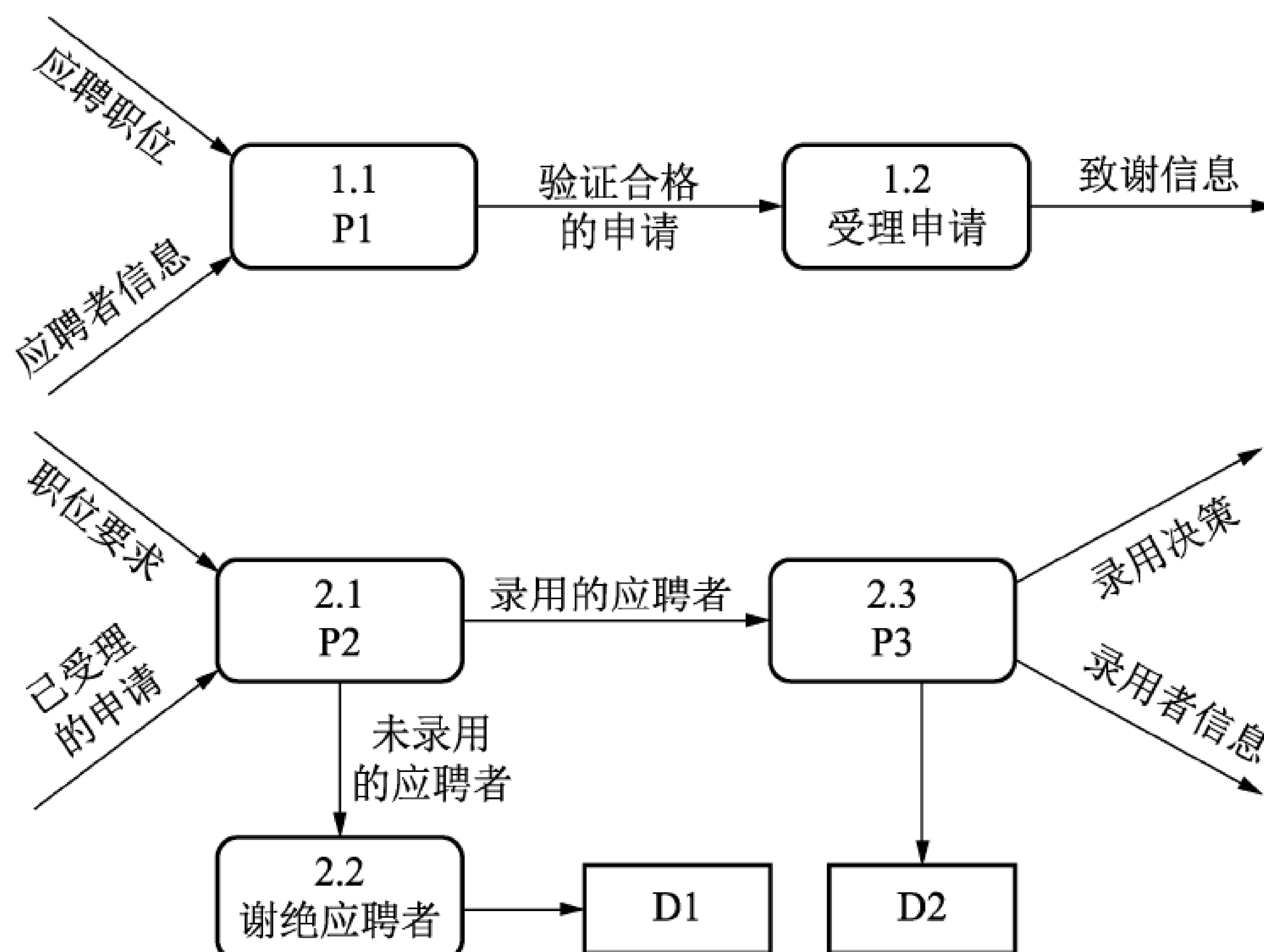


图 1-23　1 层数据流图

【问题 1】(3 分)

使用说明中的术语，给出图 1-21 中 E1~E3 所对应的外部实体名称。

【问题 2】(2 分)

使用说明中的术语，给出图 1-22 中 D1~D2 所对应的数据存储名称。

【问题 3】(6 分)

使用说明和图中的术语，给出图 1-23 中加工 P1~P3 的名称。

【问题 4】(4 分)

解释说明图 1-22 和图 1-23 是否保持平衡，若不平衡请按表 1-2 所示的格式补充图 1-23 中数据流的名称以及数据流的起点或终点，使其平衡(使用说明中的术语或图中的符号)。

表 1-2　题 3 问题 4 表

数据流名称	起　点

4. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2011 年 5 月试题一)

【说明】

某医院欲开发病人监控系统。该系统通过各种设备监控病人的生命体征，并在生命体征异常时向医生和护理人员报警。该系统的主要功能如下。

(1) 本地监控。定期获取病人的生命体征，如体温、血压、心率等数据。

(2) 格式化生命体征。对病人的各项重要生命体征数据进行格式化，然后存入日志文件并检查生命体征。

(3) 检查生命体征。将格式化后的生命体征与生命体征范围文件中预设的正常范围进行比较，如果超出了预设范围，系统就发送一条警告信息给医生和护理人员。

(4) 维护生命体征范围。医生在必要时(如新的研究结果出现时)添加或更新生命体征值的正常范围。

(5) 提取报告。在医生或护理人员请求病人生命体征报告时，从日志文件中提取病人生命体征生成体征报告，并返回给请求者。

(6) 生成病历。根据日志文件中的生命体征，医生对病人的病情进行描述，形成病历，存入病历文件。

(7) 查询病历。根据医生的病历查询请求，查询病历文件，给医生返回病历报告。

(8) 生成治疗意见。根据日志文件中的生命体征和病历，医生给出治疗意见，如处方等，并存入治疗意见文件。

(9) 查询治疗意见。医生和护理人员查询治疗意见，据此对病人进行治疗。

现采用结构化方法对病人监控系统进行分析与设计，获得如图 1-24 所示的顶层数据流图和如图 1-25 所示的 0 层数据流图。

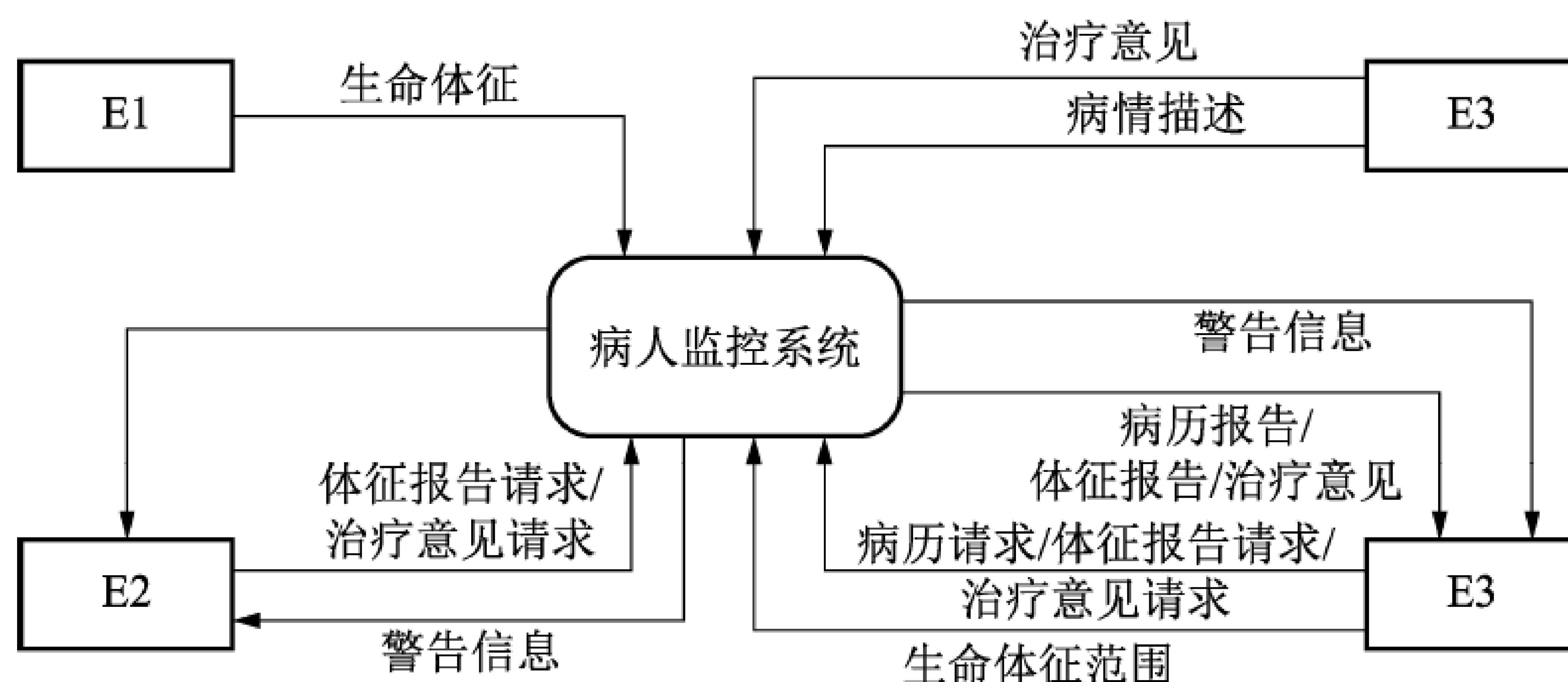


图 1-24　顶层数据流图

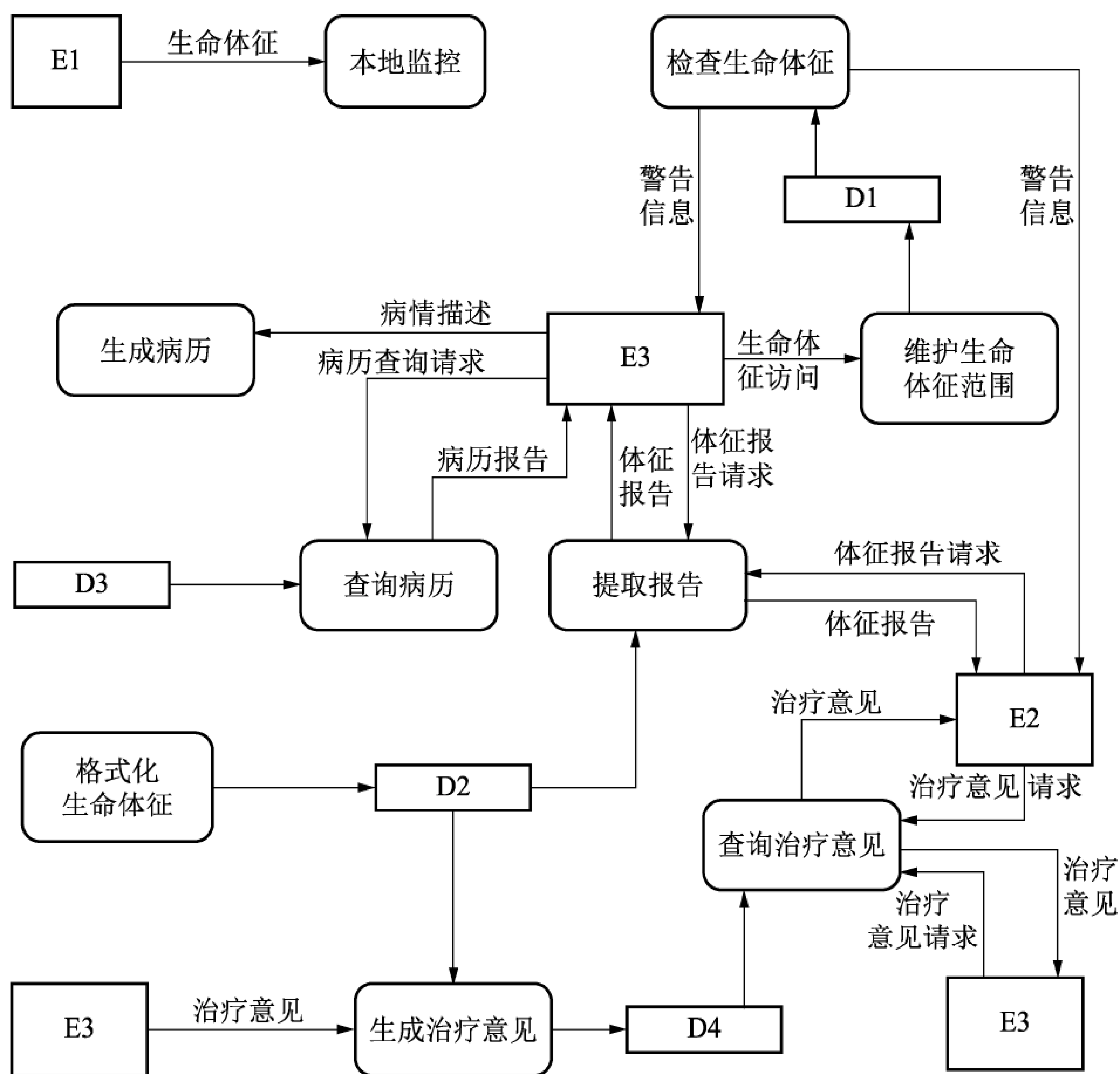


图 1-25　0 层数据流图

【问题 1】(3 分)

使用说明中的词语，给出图 1-24 中的外部实体 E1~E3 的名称。

【问题 2】(4 分)

使用说明中的词语，给出图 1-25 中的数据存储 D1~D4 的名称。

【问题 3】(6 分)

图 1-25 中缺失了 4 条数据流，使用说明、图 1-24 和图 1-25 中的术语，给出数据流的

名称及其起点和终点。

【问题 4】(3 分)

说明外部实体 E1 和 E3 之间可否有数据流，并解释其原因。

5. 阅读以下说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2010 年 11 月试题一)

【说明】

某时装邮购提供商拟开发订单处理系统，用于处理客户通过电话、传真、邮件或 Web 站点所下订单。其主要功能如下。

(1) 增加客户记录。将新客户信息添加到客户文件，并分配一个客户号以备后续使用。

(2) 查询商品信息。接收客户提交商品信息请求，从商品文件中查询商品的价格和可订购数量等商品信息，返回给客户。

(3) 增加订单记录。根据客户的订购请求及该客户记录的相关信息，产生订单并添加到订单文件中。

(4) 产生配货单。根据订单记录产生配货单，并将配货单发送给仓库进行备货；备好货后，发送备货就绪通知。如果现货不足，则需向供应商订货。

(5) 准备发货单。从订单文件中获取订单记录，从客户文件中获取客户记录，并产生发货单。

(6) 发货。当收到仓库发送的备货就绪通知后，根据发货单给客户发货；产生装运单并发送给客户。

(7) 创建客户账单。根据订单文件中的订单记录和客户文件中的客户记录，产生并发送客户账单，同时更新商品文件中的商品数量和订单文件中的订单状态。

(8) 产生应收账户。根据客户记录和订单文件中的订单信息，产生并发送给财务部门应收账户报表。

现采用结构化方法对订单处理系统进行分析与设计，获得如图 1-26 所示的顶层数据流图和如图 1-27 所示的 0 层数据流图。

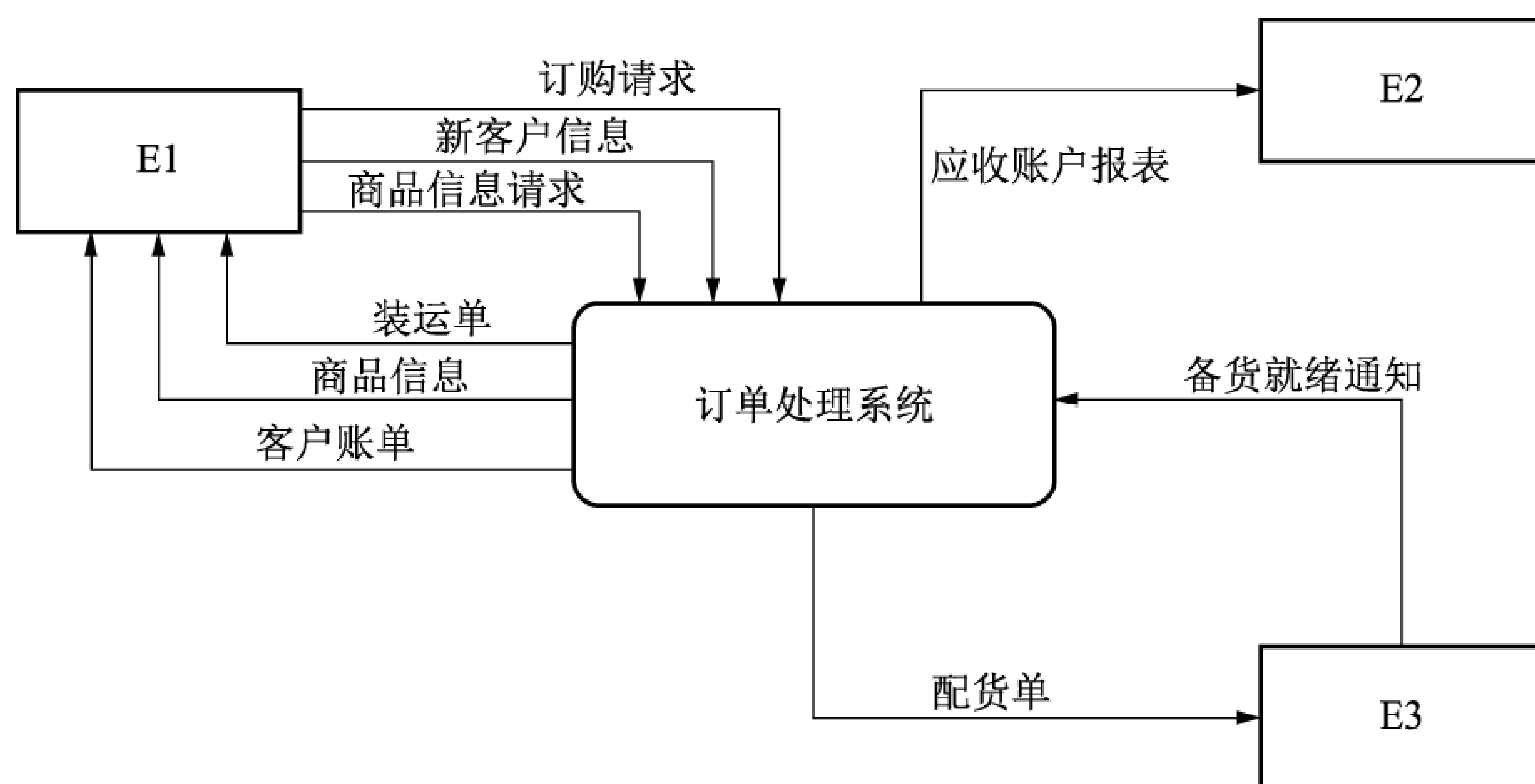


图 1-26　顶层数据流图

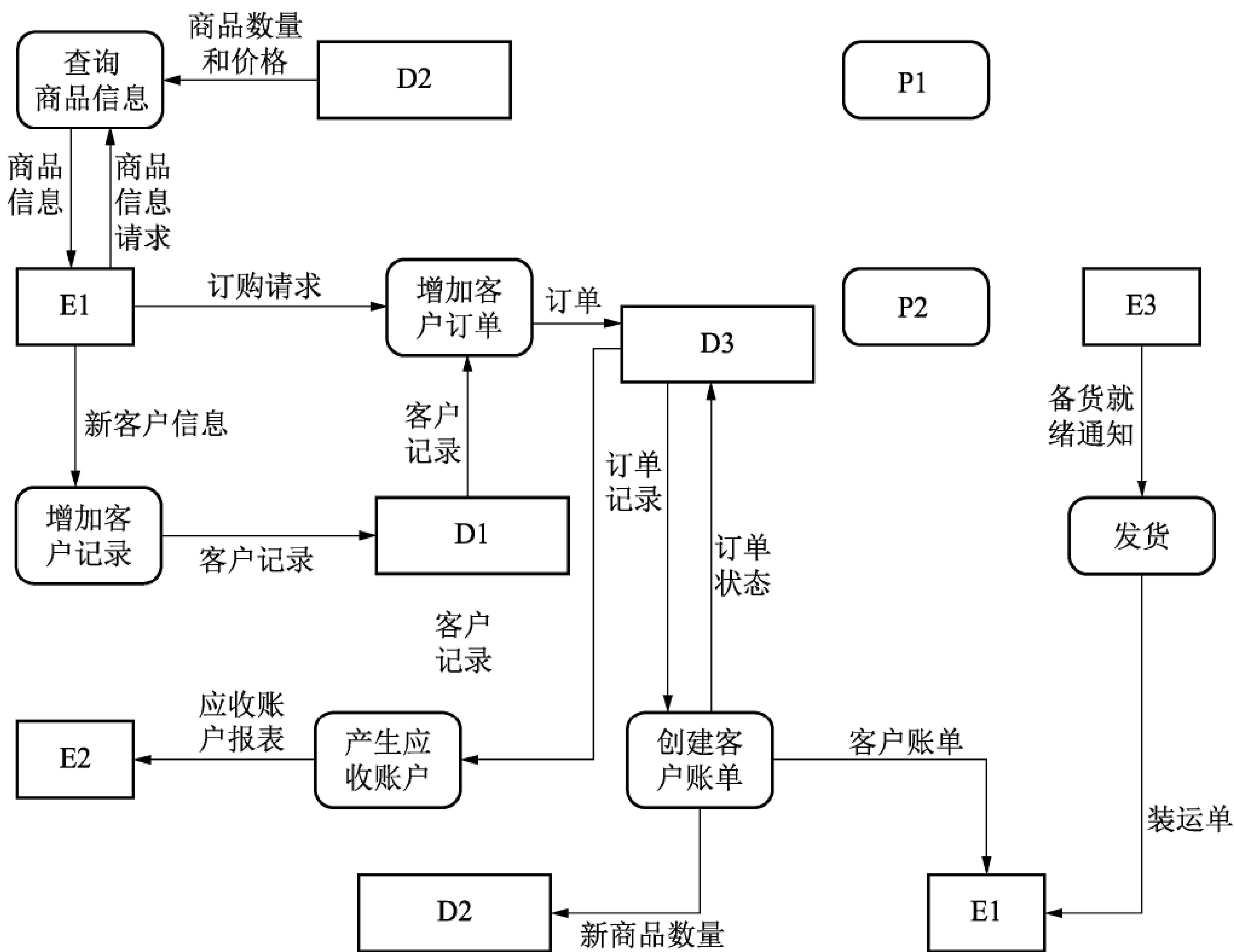


图 1-27　0 层数据流图

【问题 1】(3 分)

使用说明中的词语，给出图 1-26 中的外部实体 E1~E3 的名称。

【问题 2】(3 分)

使用说明中的词语，给出图 1-27 中的数据存储 D1~D3 的名称。

【问题 3】(9 分)

(1) 给出图 1-27 中处理(加工)P1 和 P2 的名称及其相应的输入/输出数据流。

(2) 除加工 P1 和 P2 的输入/输出数据流外，图 1-27 中还缺失了一条数据流，请给出其起点和终点，填入表 1-3 中。(注：名称使用说明中的词汇，起点和终点均使用图 1-27 中的符号或词汇。)

表 1-3　题 5 问题 3 表

起　点	终　点

6. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2010 年 5 月试题一)

【说明】

某大型企业的数据中心为了集中管理、控制用户对数据的访问并支持大量的连接需求，

欲构建数据管理中间件，其主要功能如下。

(1) 数据管理员可通过中间件进行用户管理、操作管理和权限管理。用户管理维护用户信息，用户信息(用户名、密码)存储在用户表中；操作管理维护数据实体的标准操作及其所属的后端数据库信息，标准操作和后端数据库信息存放在操作表中；权限管理维护权限表，该表存储用户可执行的操作信息。

(2) 中间件验证前端应用提供的用户信息。若验证不通过，返回非法用户信息；若验证通过，中间件将等待前端应用提交操作请求。

(3) 前端应用提交操作请求后，中间件先对请求进行格式检查。如果格式不正确，返回格式错误的信息；如果格式正确，则进行权限验证(验证用户是否有权执行请求的操作)，若用户无权执行该操作，则返回权限不足信息，否则进行连接管理。

(4) 连接管理连接相应的后端数据库并提交操作。连接管理先检查是否存在空闲的数据库连接，如果不存在，新建连接；如果存在，则重用连接。

(5) 后端数据库执行操作并将结果传给中间件，中间件对收到的操作结果进行处理后，将其返回给前端应用。

现采用结构化方法对系统进行分析与设计，获得如图 1-28 所示的顶层数据流图和如图 1-29 所示的 0 层数据流图。

【问题 1】(3 分)

使用说明中的词语，给出图 1-28 中的外部实体 E1~E3 的名称。

【问题 2】(3 分)

使用说明中的词语，给出图 1-29 中的数据存储 D1~D3 的名称。

【问题 3】(6 分)

给出图 1-29 中加工 P 的名称及其输入/输出数据流，填入表 1-4 中。

除加工 P 的输入/输出数据流外，图 1-29 还缺失了两条数据流，请给出这两条数据流的起点和终点，填入表 1-5 中。(注：名称使用说明中的词汇，起点和终点均使用图 1-29 中的符号或词汇。)

【问题 4】(3 分)

在绘制数据流图时，需要注意加工的绘制。请给出三种在绘制加工的输入、输出时可能出现的错误。

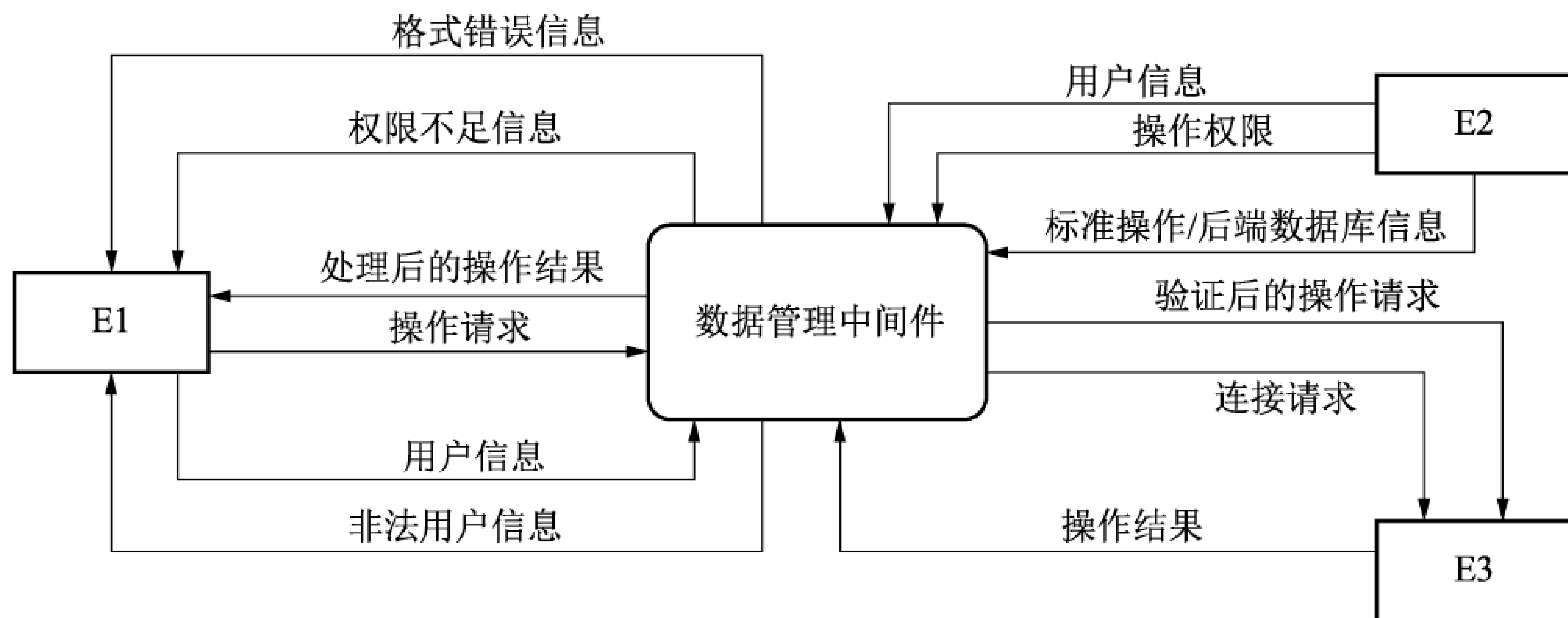


图 1-28　顶层数据流图

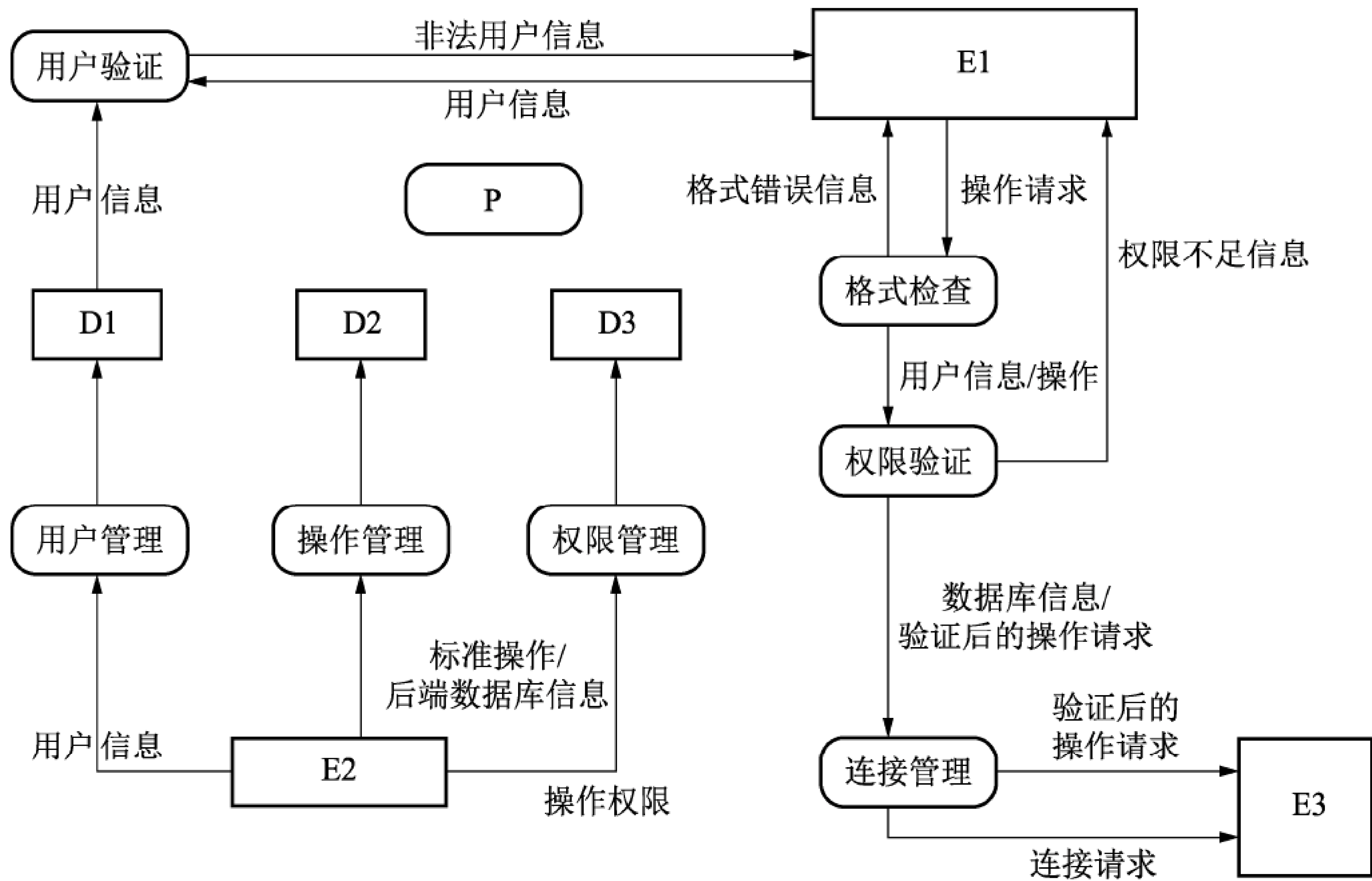


图 1-29　0 层数据流图

表 1-4　题 6 问题 3 表(1)

	名　称	起　点	终　点
输入数据流			P
输出数据流		P	

表 1-5　题 6 问题 3 表(2)

起　点	终　点

7. 阅读下列说明，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2009 年 11 月试题一)

【说明】

现准备为某银行开发一个信用卡管理系统(CCMS)，该系统的基本功能如下。

(1) 信用卡申请。非信用卡客户填写信用卡申请表，说明所要申请的信用卡类型及申请者的基本信息，提交 CCMS。如果信用卡申请被银行接受，CCMS 将记录该客户的基本信息，并发送确认函给该客户，告知客户信用卡的有效期及信贷限额；否则该客户将会收到一封拒绝函。非信用卡客户收到确认函后成为信用卡客户。

(2) 信用卡激活。信用卡客户向 CCMS 提交激活请求，用信用卡号和密码激活该信用卡。激活操作结束后，CCMS 将激活通知发送给客户，告知客户其信用卡是否被成功激活。

(3) 信用卡客户信息管理。信用卡客户的个人信息可以在 CCMS 中进行在线管理。每位信用卡客户可以在线查询和修改个人信息。

(4) 交易信息查询。信用卡客户使用信用卡进行的每一笔交易都会记录在 CCMS 中。信用卡客户可以通过 CCMS 查询并核实其交易信息(包括信用卡交易记录及交易额)。

图 1-30 和图 1-31 分别给出了该系统的顶层数据流图和 0 层数据流图。

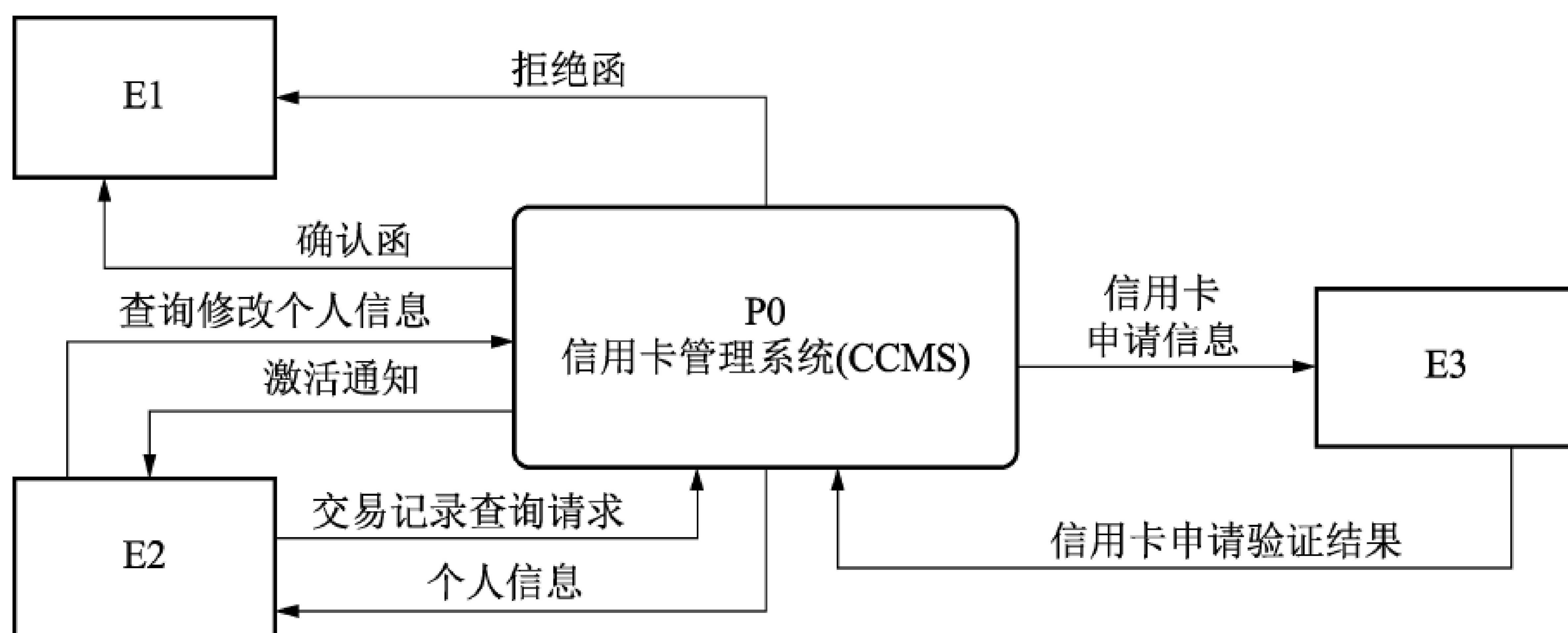


图 1-30　顶层数据流图

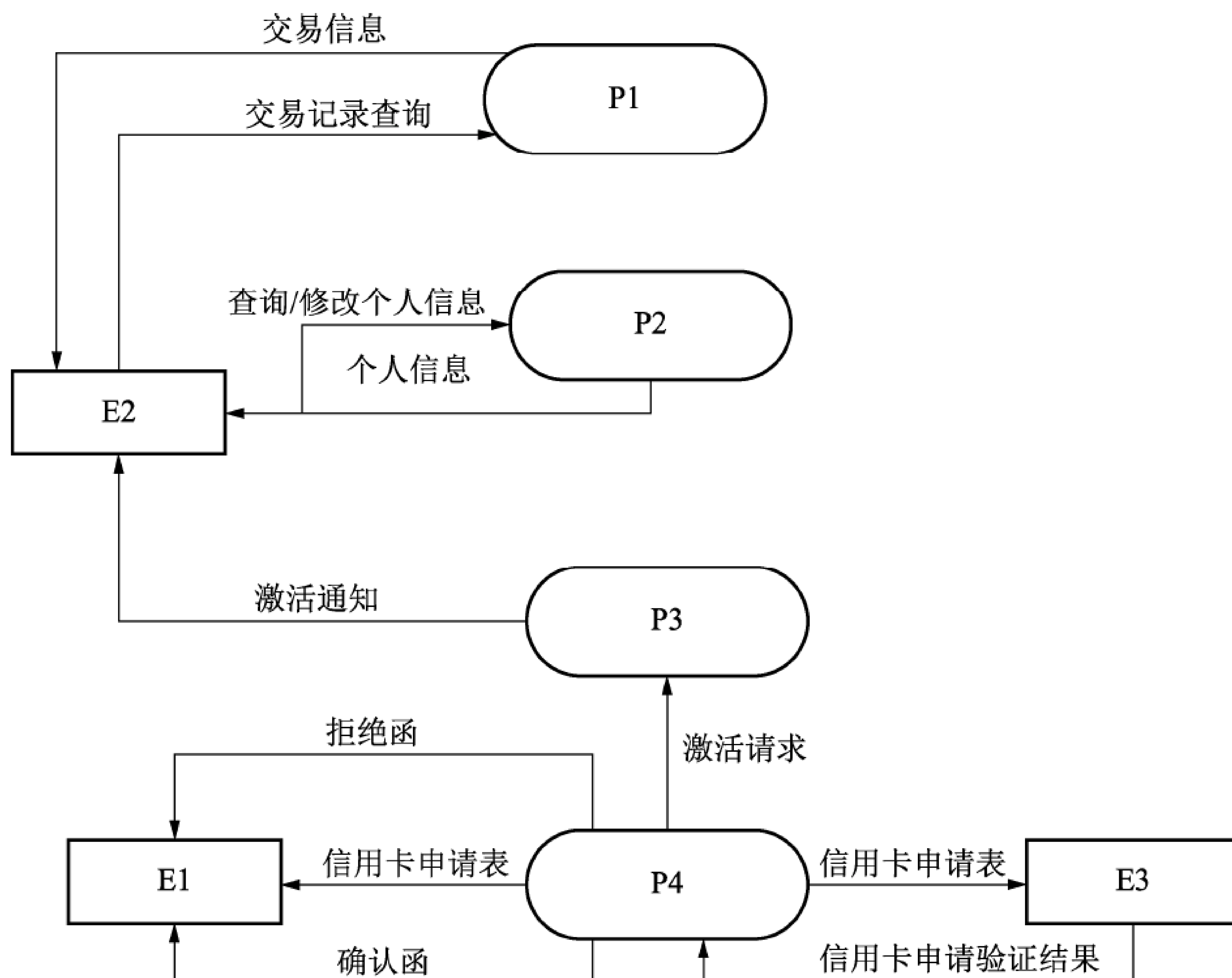


图 1-31　0 层数据流图

【问题 1】

根据说明，将图 1-30 中的 E1~E3 填充完整。

【问题 2】

图 1-30 中缺少 3 条数据流，根据说明分别指出这 3 条数据流的起点和终点。(注：数据

流的起点和终点均采用图中的符号和描述。)

【问题 3】

图 1-29 中有两条数据流是错误的，请指出这两条数据流的名称并改正。(注：数据流的起点和终点均采用图中的符号和描述。)

【问题 4】

根据说明，将图 1-31 中 P1~P4 处的名称填充完整。

8. 阅读下列说明，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。(2009 年 5 月试题一)

【说明】

假设某大型商业企业由商品配送中心和连锁超市组成，其中商品配送中心包括采购、财务、配送等部门。为了实现高效管理，设计了商品配送中心信息管理系统，其主要功能描述如下。

(1) 系统接受由连锁超市提出的供货请求，并将其记录到供货请求记录文件。

(2) 在接到供货请求后，从商品库存记录文件中进行商品库存信息查询。如果库存满足供货请求，则给配送处理发送配送通知；否则，向采购部门发出缺货通知。

(3) 配送处理接到配送通知后，查询供货请求记录文件，更新商品库存记录文件，并向配送部门发送配送单，在配送货品的同时记录配送信息至商品配送记录文件。

(4) 采购部门接到缺货通知后，与供货商洽谈，进行商品采购处理，合格商品入库，并将采购清单记录至采购清单记录文件，向配送处理发出配送通知，同时通知财务部门给供货商支付货款。

该系统采用结构化方法进行开发，得到待修改的数据流图如图 1-32 所示。

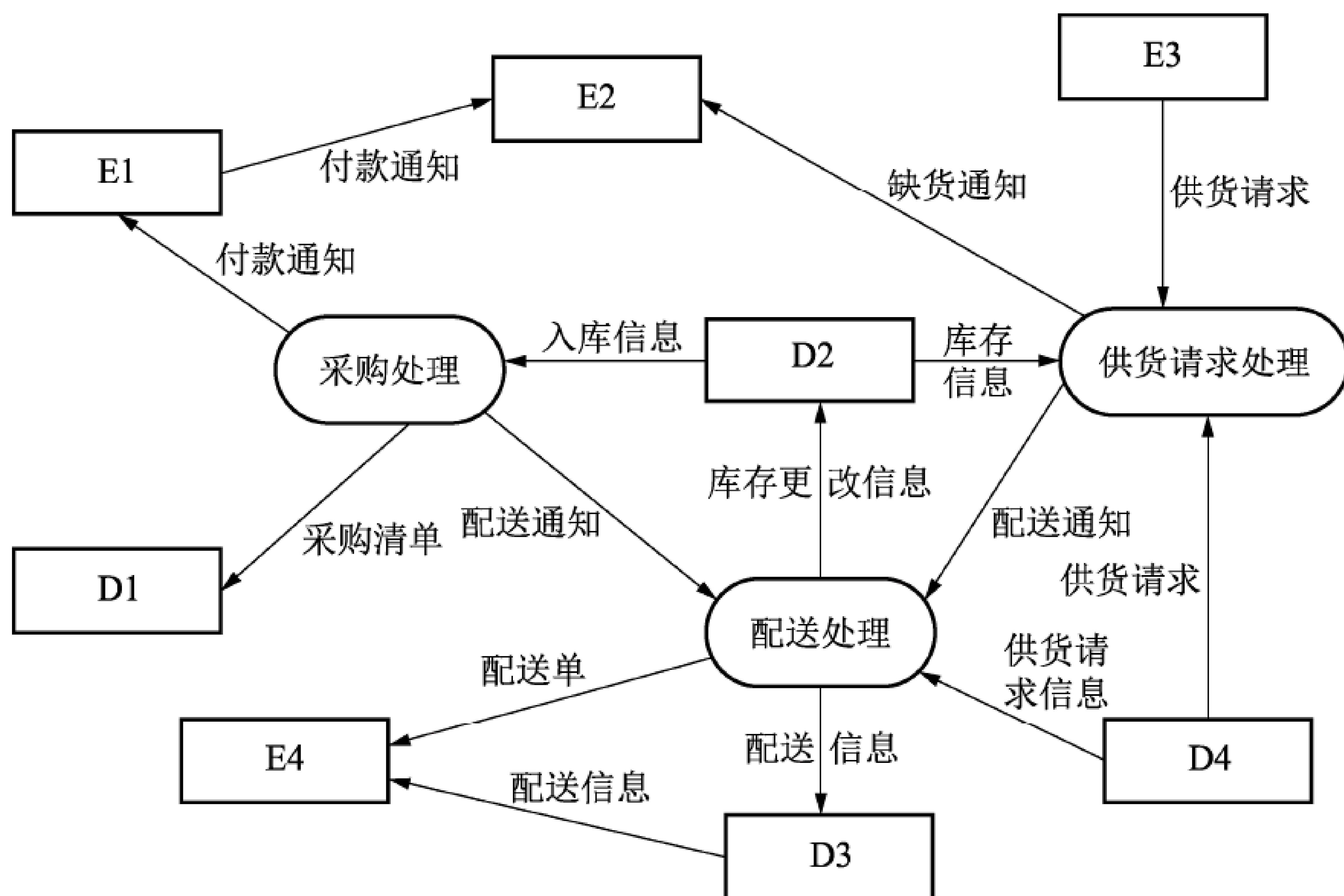


图 1-32 数据流图

【问题 1】

使用说明中的词语，给出图 1-32 中外部实体 E1~E4 的名称和数据存储 D1~D4 的名称。

【问题 2】

图 1-32 中存在 4 处错误数据流，请指出各自的起点和终点，填入表 1-6 中；若将上述 4 条错误数据流删除，为保证数据流图的正确性，应补充 3 条数据流，请给出所补充数据流的起点和终点，填入表 1-7 中。(注：起点和终点采用图 1-32 中的符号或名称。)

表 1-6　题 8 问题 2 表(1)

起　点	终　点

表 1-7　题 8 问题 2 表(2)

起　点	终　点

9. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2008 年 11 月试题一)

【说明】

某营销企业拟开发一个销售管理系统，其主要功能描述如下。

(1) 接受客户订单，检查库存货物是否满足订单要求。如果满足，进行供货处理，即修改库存记录文件，给库房开具备货单并且保留客户订单至订单记录文件；否则进行缺货处理，即将缺货记录单存入缺货记录文件。

(2) 根据缺货记录文件进行缺货统计，将缺货通知单发给采购部门。

(3) 根据采购部门提供的进货通知单进行进货处理，即修改库存记录文件，并从缺货记录文件中取出缺货订单进行供货处理。

(4) 根据保留的客户订单进行销售统计，打印统计报表给经理。

现采用结构化方法对销售管理系统进行分析与设计，获得如图 1-33 所示的顶层数据流图和如图 1-34 所示的 0 层数据流图。

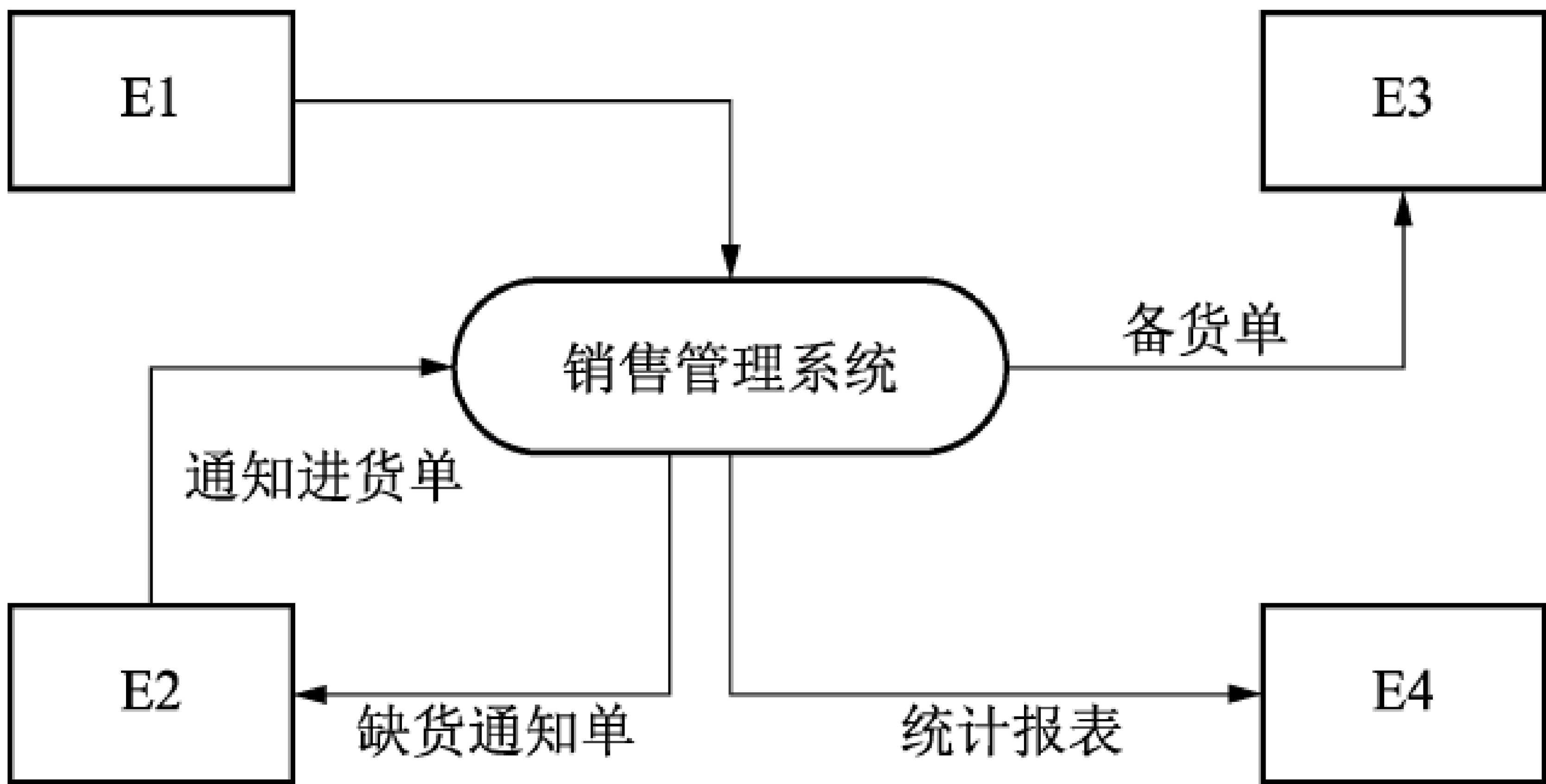


图 1-33　顶层数据流图

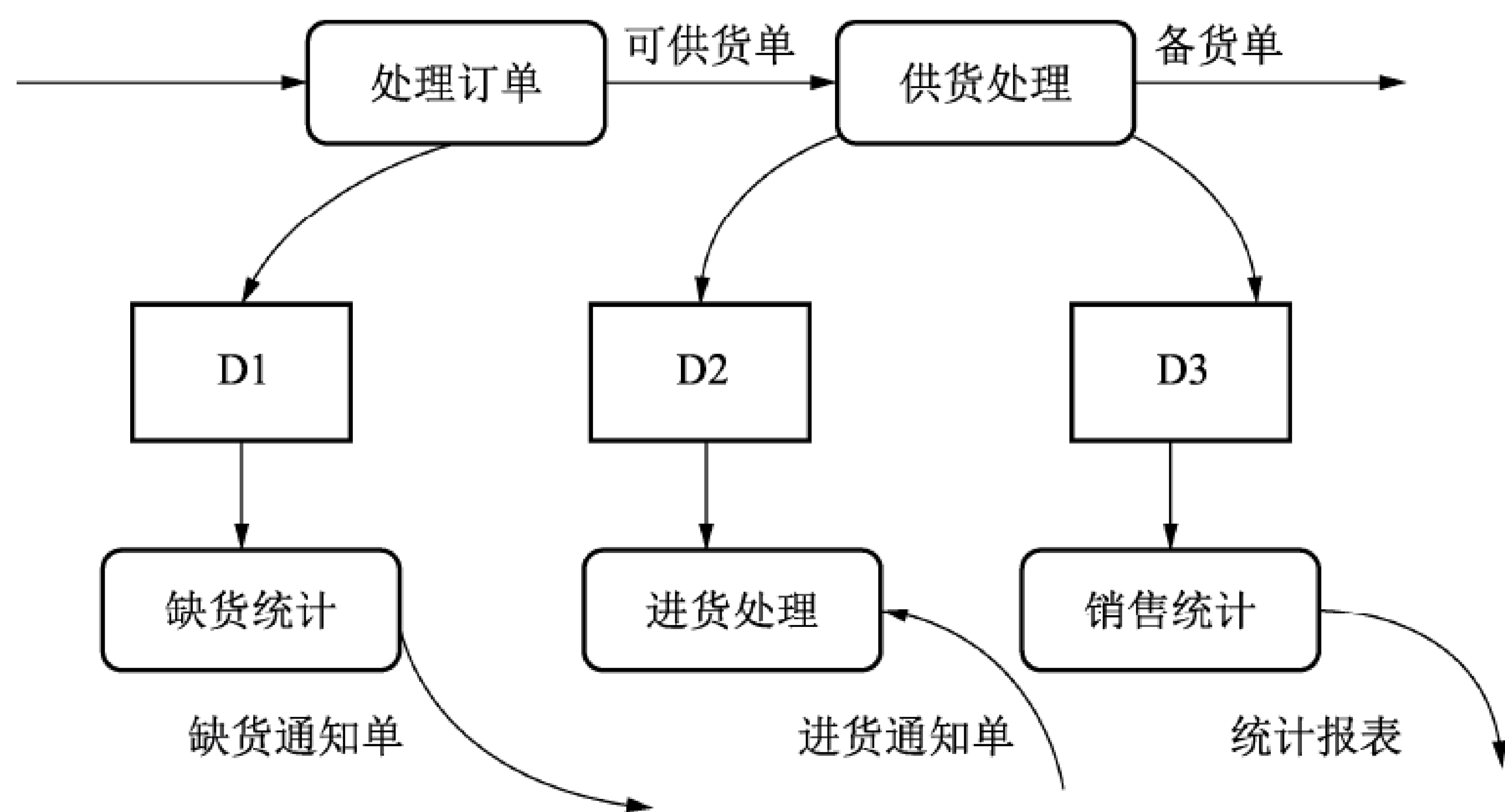


图 1-34　0 层数据流图

【问题 1】

使用说明中的词语，给出图 1-33 所示的外部实体 E1~E4 的名称。

【问题 2】

使用说明中的词语，给出图 1-34 所示的数据存储 D1~D3 的名称。

【问题 3】

0 层数据流图 1-34 中缺少了 4 条数据流，根据说明及顶层数据流图 1-33 所提供的信息，分别指出这 4 条数据流的起点和终点，填入表 1-8 中。

表 1-8　题 9 问题 3 表

起　点	终　点

10. 阅读以下说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2008 年 5 月试题一)

【说明】

某音像制品出租商店欲开发一个音像管理信息系统，管理音像制品的租借业务，需求如下。

(1) 系统中的客户信息文件保存了该商店所有客户的用户名、密码等信息。对于首次来租借的客户，系统会为其生成用户名和初始密码。

(2) 系统中音像制品信息文件记录了商店中所有音像制品的详细信息及库存数量。

(3) 根据客户所租借的音像制品的品种，会按天收取相应的费用。音像制品的最长租借周期为一周，每位客户每次最多只能租借 6 件音像制品。

(4) 客户租借某种音像制品的具体流程如下。

① 根据客户提供的用户名和密码，验证客户身份。

② 若该客户是合法客户，查询音像制品信息文件，查看商店中是否还有这种音像制品。

③ 若还有该音像制品，且客户所要租借的音像制品数不多于 6 件，就可以将该音像制品租借给客户。这时，系统给出相应的租借确认信息，生成一条新的租借记录并将其保存在租借记录文件中。

④ 系统计算租借费用，将费用信息保存在租借记录文件中并告知客户。

⑤ 客户付清租借费用之后，系统接收客户付款信息，将音像制品租借给该客户。

(5) 当库存中某音像制品数量不能满足客户的租借请求数量时，系统可以接受客户网上预约租借某种音像制品。系统接收到预约请求后，检查库存信息，验证用户身份，创建相应的预约记录，生成预约流水号给该客户，并将信息保存在预约记录文件中。

(6) 客户归还到期的音像制品，系统修改租借记录文件，并查阅预约记录文件和客户信息文件，判定是否有客户预约了这些音像制品。若有，则生成预约提示信息，通知系统履行预约服务，系统查询客户信息文件和预约记录文件，通知相关客户前来租借音像制品。

现采用结构化方法对音像管理信息系统进行分析与设计，得到如图 1-35 所示的顶层数据流图和如图 1-36 所示的 0 层数据流图。

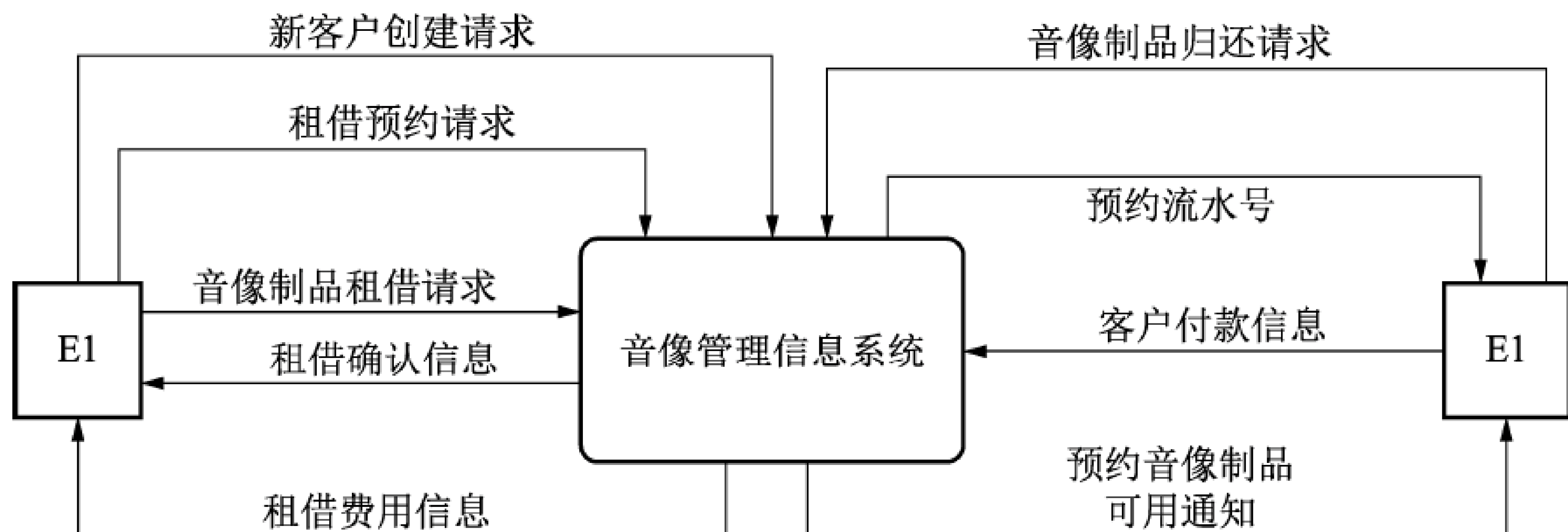


图 1-35　顶层数据流图

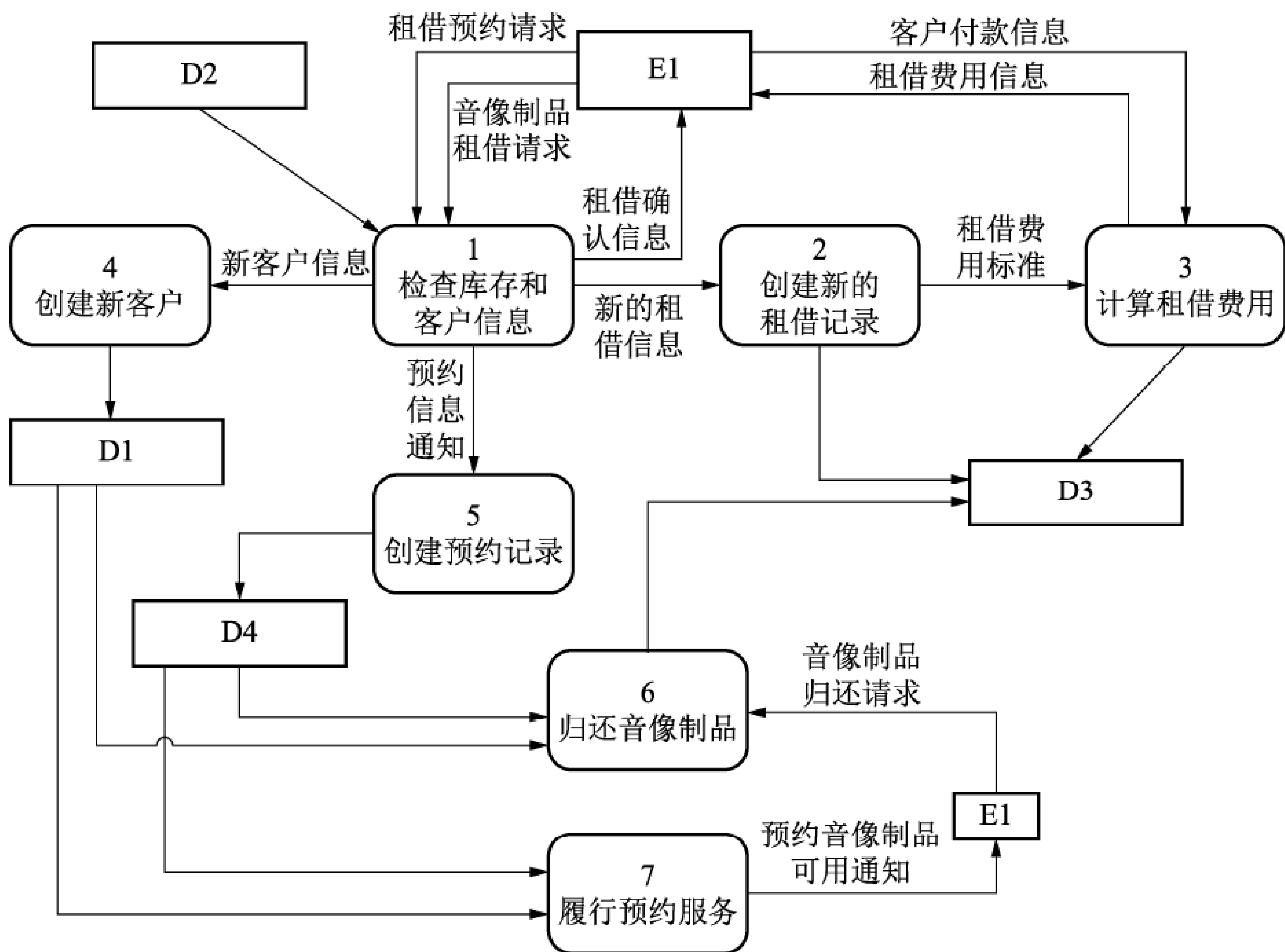


图 1-36　0 层数据流图

【问题 1】

图 1-35 中只有一个外部实体 E1。使用说明中的词语，给出 E1 的名称。

【问题 2】

使用说明中的词语，给出图 1-36 中数据存储 D1~D4 的名称。

【问题 3】

图 1-36 中缺少了 3 条数据流，根据说明及数据流图 1-35 所提供的信息，分别指出这 3 条数据流的起点和终点，填入表 1-9 中。

表 1-9　题 10 问题 3 表

起　点	终　点

【问题 4】

在进行系统分析与设计时，面向数据结构的设计方法(如 Jackson 方法)也被广泛应用。简要说明面向数据结构的设计方法的基本思想及其适用场合。

11. 阅读以下说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2007 年 11 月试题一)

【说明】

某高校欲开发一个成绩管理系统，记录并管理所有选修课程的学生的平时成绩和考试成绩，其主要功能描述如下。

(1) 每门课程都由 3~6 个单元构成，每个单元结束后会进行一次测试，其成绩作为这门课程的平时成绩。课程结束后进行期末考试，其成绩作为这门课程的考试成绩。

(2) 学生的平时成绩和考试成绩均由每门课程的主讲教师上传给成绩管理系统。

(3) 在记录学生成绩之前，系统需要验证这些成绩是否有效。首先，根据学生信息文件来确认该学生是否选修这门课程，若没有，那么这些成绩是无效的；如果学生的确选修了这门课程，再根据课程信息文件和课程单元信息文件来验证平时成绩是否与这门课程所包含的单元相对应，如果是，那么这些成绩是有效的，否则无效。

(4) 对于有效成绩，系统将其保存在课程成绩文件中。对于无效成绩，系统会单独将其保存在无效成绩文件中，并将详细情况提交给教务处。在教务处没有给出具体处理意见之前，系统不会处理这些成绩。

(5) 若一门课程的所有有效的平时成绩和考试成绩都已经被系统记录，系统会发送课程完成通知给教务处，告知该门课程的成绩已经齐全。教务处根据需要，请求系统生成相应的成绩列表，用来提交考试委员会审查。

(6) 在生成成绩列表之前，系统会生成一份成绩报告给主讲教师，以便核对是否存在错误。主讲教师须将核对之后的成绩报告返还系统。

(7) 根据主讲教师核对后的成绩报告，系统生成相应的成绩列表，递交考试委员会进行审查。考试委员会在审查之后，上交一份成绩审查结果给系统。对于所有通过审查的成绩，

系统将会生成最终的成绩单，并通知每个选课学生。

现采用结构化方法对这个系统进行分析与设计，得到如图 1-37 所示的顶层数据流图和如图 1-38 所示的 0 层数据流图。

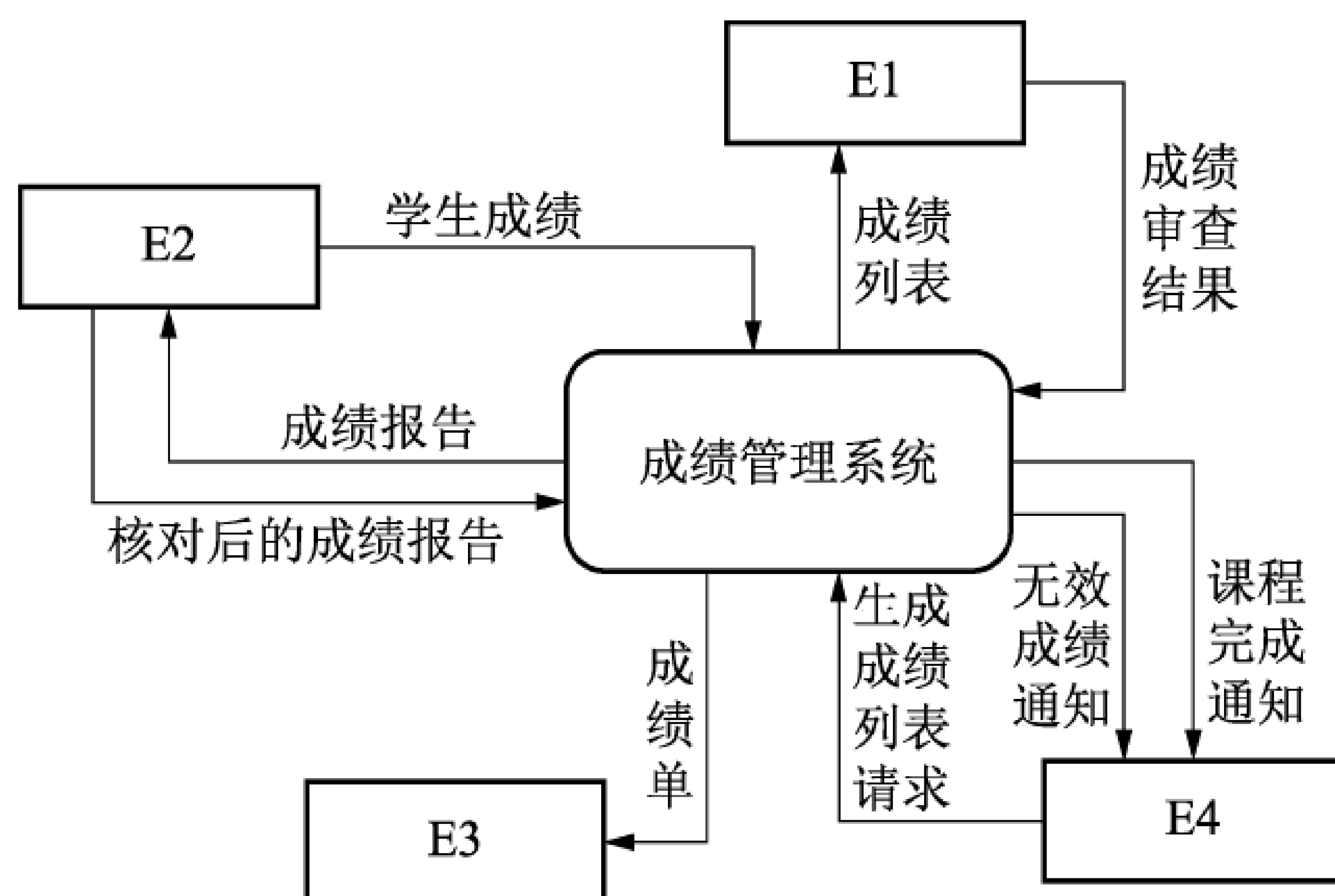


图 1-37　顶层数据流图

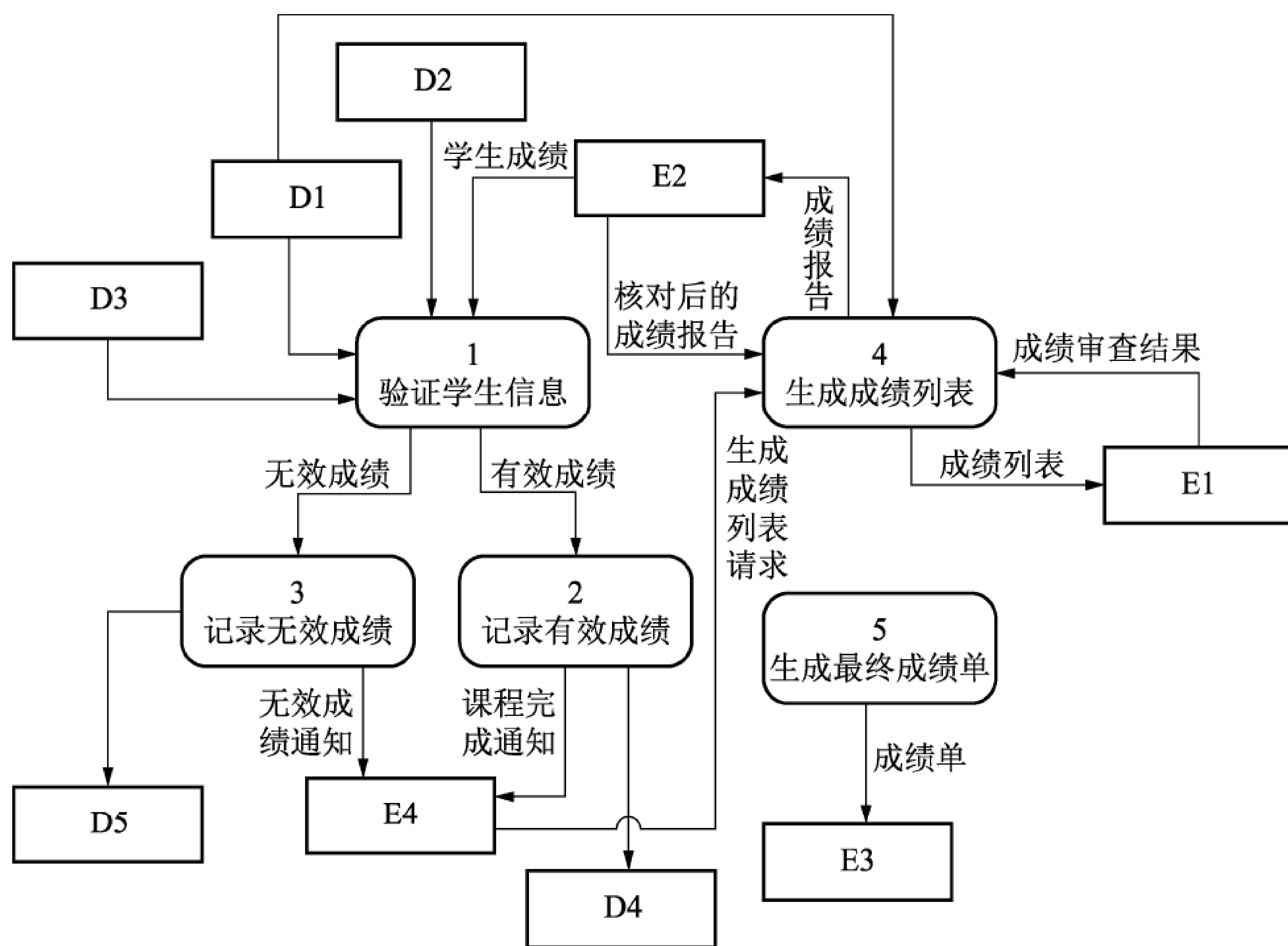


图 1-38　0 层数据流图

【问题 1】

使用说明中的词语，给出图 1-37 中外部实体 E1~E4 的名称。

【问题 2】

使用说明中的词语，给出图 1-38 中数据存储 D1~D5 的名称。

【问题3】

图1-38中缺少了3条数据流，根据说明及图1-37所提供的信息，分别指出这3条数据流的起点和终点，填入表1-10中。

表1-10　题11问题3表

起　点	终　点

【问题4】

数据流图是在系统分析与总体设计阶段宏观地描述系统功能需求的重要图形化工具，程序流程图也是软件开发过程中比较常用的图形化工具。简要说明程序流程图的适用场合与作用。

12. 阅读以下说明和图，回答问题1~问题3，将解答填入答题纸的对应栏内。(2007年5月试题一)

【说明】

某房屋租赁公司欲建立一个房屋租赁服务系统，统一管理房主和租赁者的信息，从而快速地提供租赁服务。该系统具有以下功能。

(1) 登记房主信息。对于每名房主，系统需登记其姓名、住址和联系电话，并将这些信息写入房主信息文件。

(2) 登记房屋信息。所有在系统中登记的房屋都有一个唯一的识别号(对于新增加的房屋，系统会自动为其分配一个识别号)。除此之外，还需登记该房屋的地址、房型(如平房、带阳台的楼房、独立式住宅等)、最多能够容纳的房客数、租金及房屋状况(待租赁、已出租)。这些信息都保存在房屋信息文件中。一名房主可以在系统中登记多个待租赁的房屋。

(3) 登记租赁者信息。所有想通过该系统租赁房屋的租赁者，必须首先在系统中登记个人信息，包括姓名、住址、电话号码、出生年月和性别。这些信息都保存在租赁者信息文件中。

(4) 租赁房屋。已经登记在系统中的租赁者可以得到一份系统提供的待租赁房屋列表。一旦租赁者从中找到合适的房屋，就可以提出看房请求。系统会安排租赁者与房主见面。对于每次看房，系统会生成一条看房记录并将其写入看房记录文件中。

(5) 收取手续费。房主登记完房屋后，系统会生成一份费用单，房主根据费用单缴纳相应的费用。

(6) 变更房屋状态。当租赁者与房主达成租房或退房协议后，房主向系统提交变更房屋状态的请求。系统将根据房主的请求，修改房屋信息文件。

图1-39和图1-40分别给出了该系统的顶层数据流图和0层数据流图。

【问题1】

使用说明中给出的词汇，将图1-39中(1)~(4)处的数据流补充完整。

【问题2】

使用说明中给出的词汇，将图1-40中的(5)~(8)补充完整。

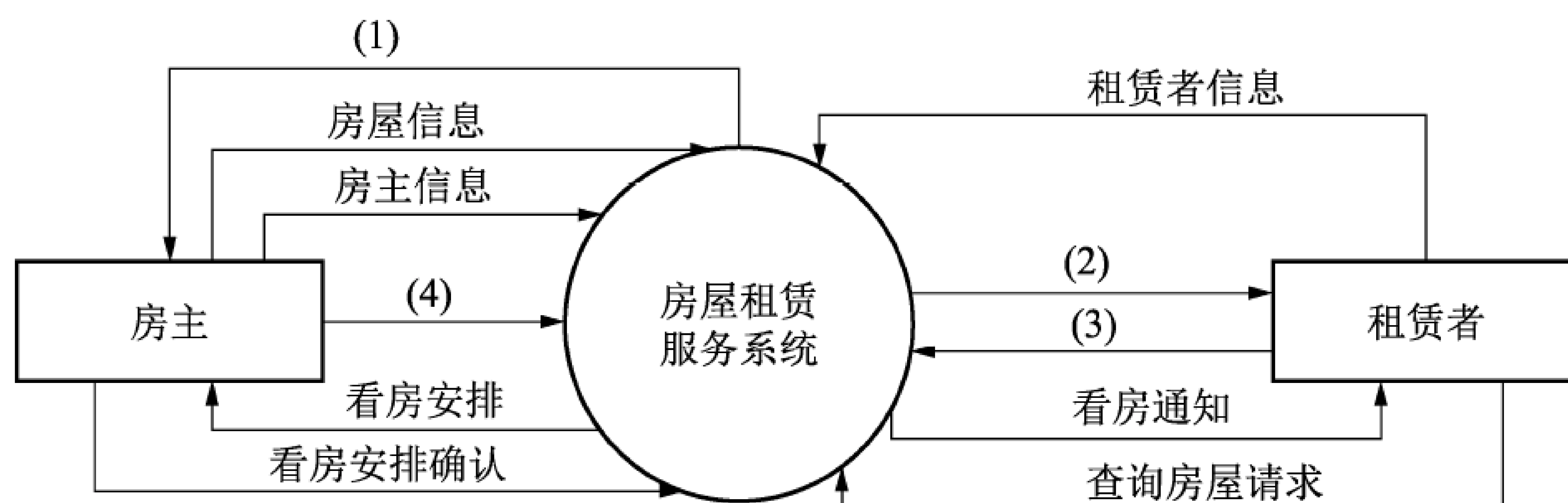


图 1-39　顶层数据流图

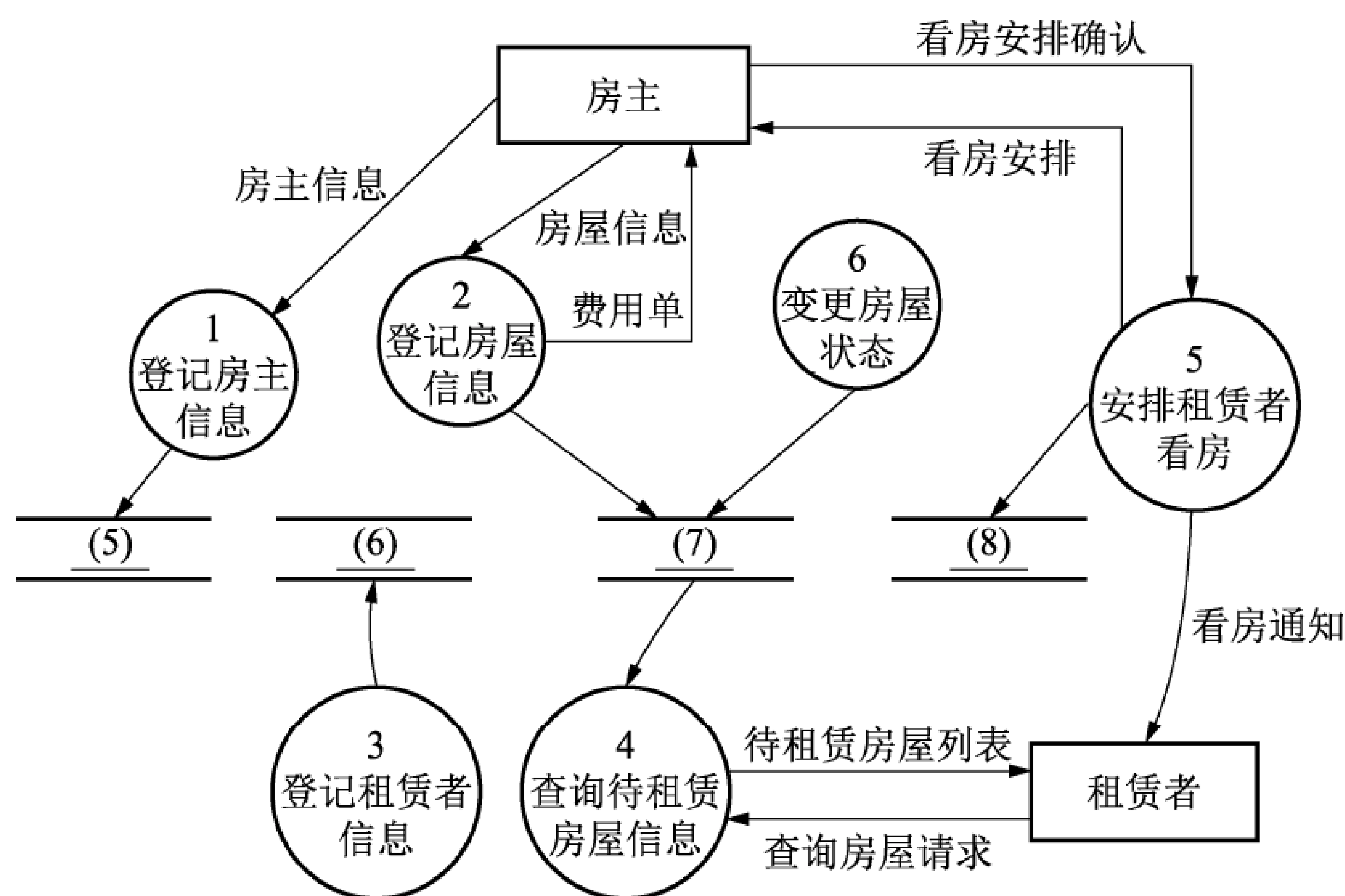


图 1-40　0 层数据流图

【问题 3】

图 1-40 中缺失了 3 条数据流，请指出这 3 条数据流的起点、终点和数据流名称。

13. 阅读以下说明以及数据流图，回答问题 1~问题 5，将解答填入答题纸的对应栏内。(2006 年 11 月试题一)

【说明】

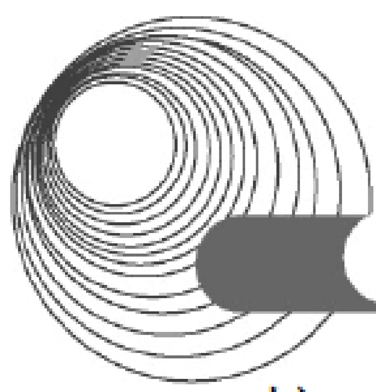
某银行已有一套基于客户机/服务器模式的储蓄系统 A 和一套建账软件。建账软件主要用于将储蓄所手工处理的原始数据转换为储蓄系统 A 所需的数据格式。该建账软件具有以下功能。

(1) 分户账录入。手工办理业务时建立的每个分户账数据均由初录员和复录员分别录入，以确保数据的正确性。

(2) 初录/复录比对。将初录员和复录员录入的数据进行一一比较，并标记两套数据是否一致。

(3) 数据确认。当上述两套数据完全一致后，将其中任一套作为最终进入储蓄系统 A 的原始数据。

(4) 汇总、核对和打印。对经过确认的数据进行汇总，并和会计账目中的相关数据进行



核对，以确保数据的整体正确性。打印输出经过确认的数据，为以后核查可能的错误提供依据。

(5) 数据转换。将经过确认的数据转换为储蓄系统 A 需要的中间格式数据。

(6) 数据清除。为加快初录和复录的处理速度，在数据确认之后，可以有选择地清除初录员和复录员录入的数据。

该软件的数据流图如图 1-41~图 1-43 所示。图中部分数据流数据文件的格式如下。

初录分户账=储蓄所号+账号+户名+开户日+开户金额+当前余额+性质

复录分户账=储蓄所号+账号+户名+开户日+开户金额+当前余额+性质

初录数据=手工分户账+一致性标志

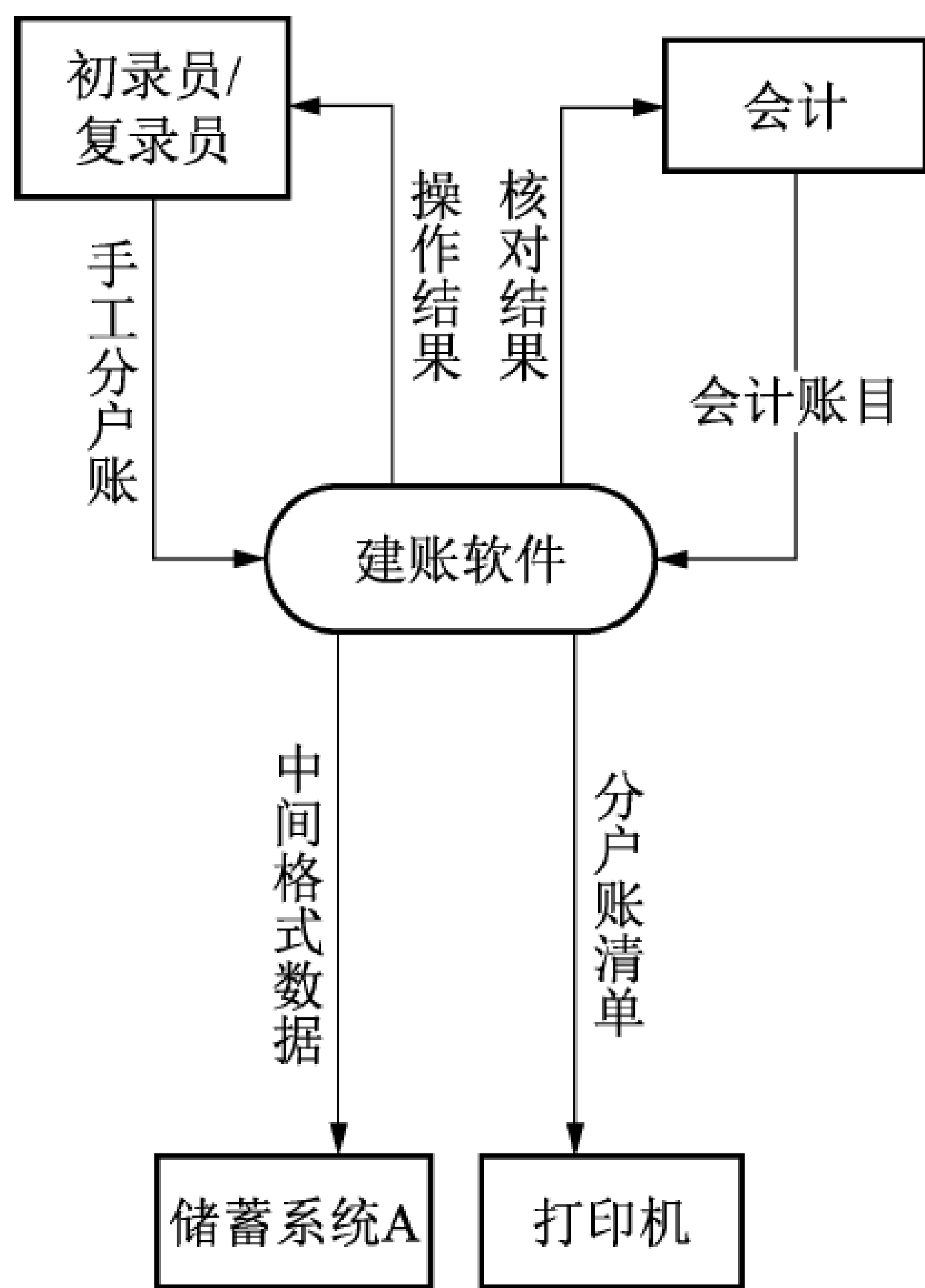


图 1-41 建账软件的顶层数据流图

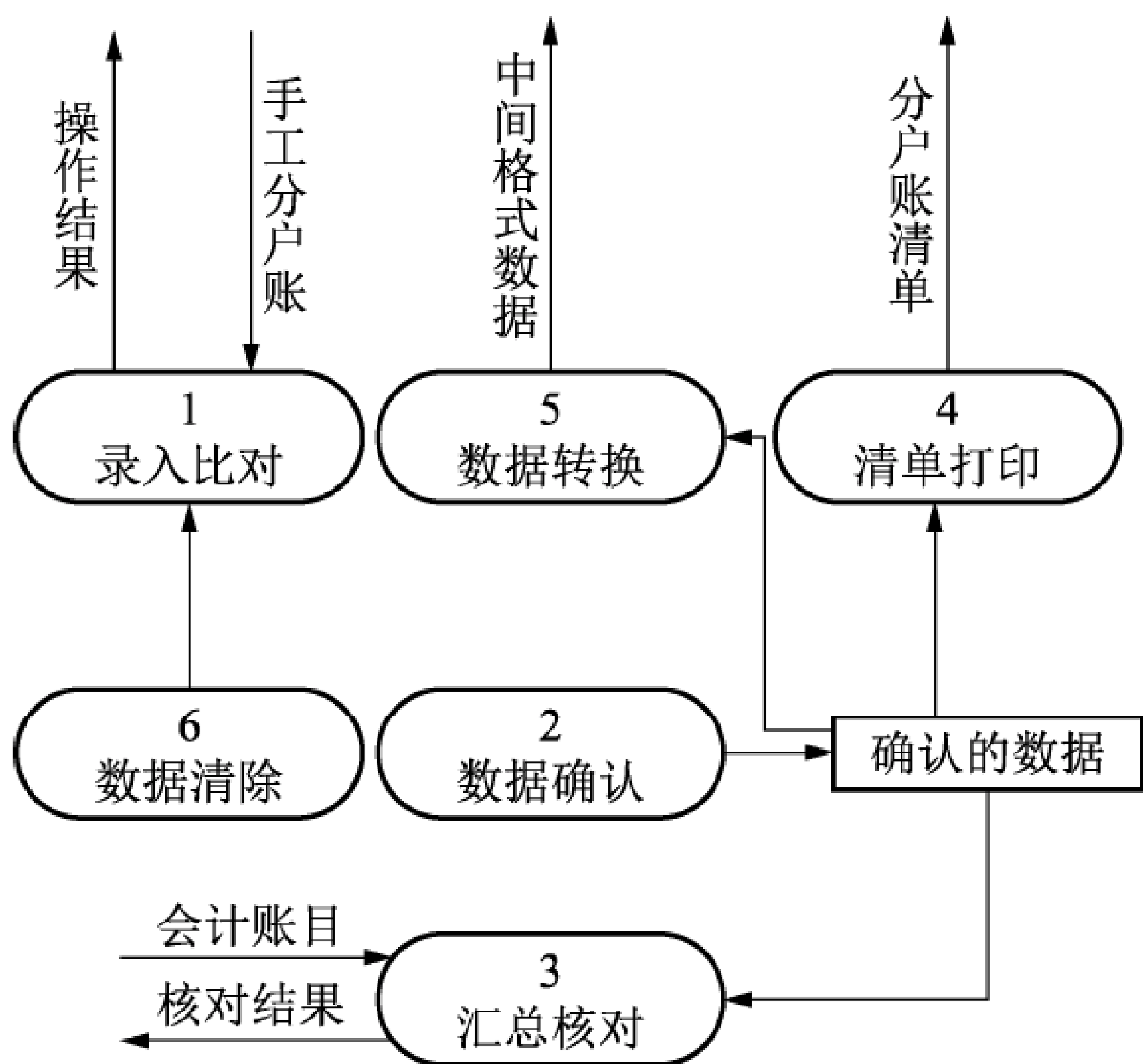


图 1-42 建账软件的 0 层数据流图

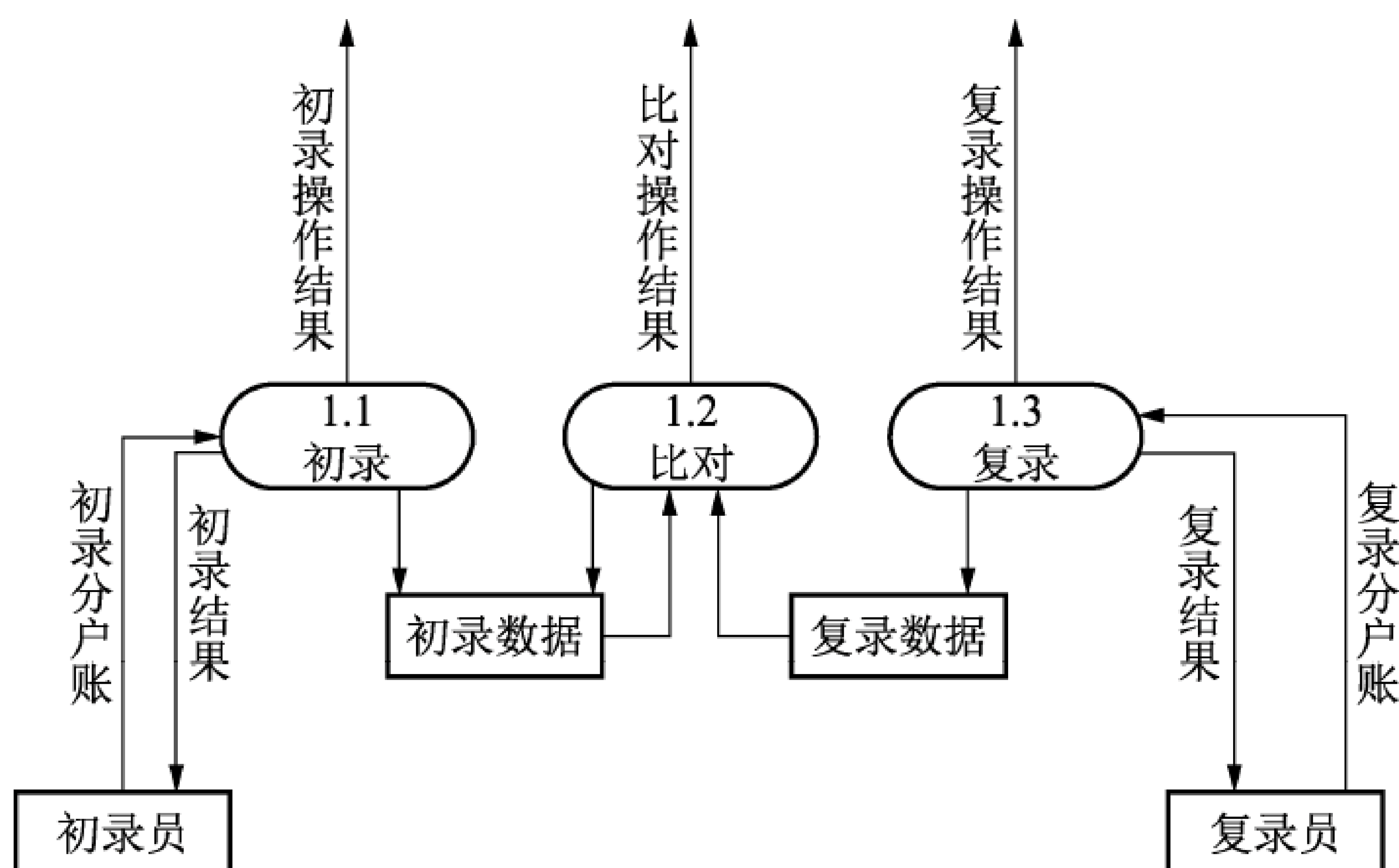


图 1-43　建账软件的 1 层数据流图

复录数据=手工分户账+一致性标志

会计账目=储蓄所号+总户数+总余额

操作结果=初录操作结果+比对操作结果+复录操作结果

软件需要打印的分户账清单样式如表 1-11 所示。

表 1-11　分户账清单样式表

储 蓄 所	账　号	开 户 日	户　名	其他分户账数据
储蓄所 1				
	…	…	…	…
储蓄所 1 合计	共×××户，总余额 9999999.99 元			
储蓄所 2				
	…	…	…	…
储蓄所 2 合计	共×××户，总余额 9999999.99 元			

【问题 1】

请采用说明中的词汇，给出数据确认处理所需的数据流在 1 层数据流图中的全部可选起点(0 层数据流图和 1 层数据流图中均未给出)。

【问题 2】

不考虑数据确认处理(加工 2)，请指出数据流图中存在的错误。

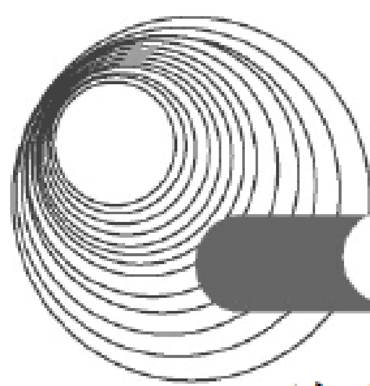
【问题 3】

打印分户账清单时，必须以下列哪一组数据作为关键字进行排序才能满足需求？请从下面选项中选择，并将对应序号填入答题纸的相应栏内。

① 储蓄所　② 账号　③ 开户日　④ 总户数和总余额

【问题 4】

加工 1(录入比对处理)除能够检查出初录数据和复录数据的不一致外，还应当检测出下



列哪些错误？请将对应序号填入答题纸的相应栏内。

① 输入的无效字符　② 输入的半个汉字　③ 显示器无法显示

④ 初录员重复录入同一账户　⑤ 汇总数据与会计账目不符　⑥ 打印机卡纸

【问题 5】

请使用数据字典条目定义形式，给出 0 层 DFD 中的“手工分户账”数据流和 1 层 DFD 中的“初录分户账”“复录分户账”的关系。

14. 阅读下列说明及数据流图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2006 年 5 月试题一)

【说明】

某学校建立了一个网上作业提交与管理系统，基本功能描述如下。

(1) 账号和密码。任课老师用账号和密码登录系统后，提交所有选课学生的名单。系统自动为每个选课学生创建登录系统的账号和密码。

(2) 作业提交。学生使用账号和密码登录系统后，可以向系统申请所选课程的作业。系统首先检查学生的当前状态，如果该学生还没有做过作业，则从数据库服务器申请一份作业。若申请成功，则显示需要完成的作业。学生需在线完成作业，单击“提交”按钮上交作业。

(3) 在线批阅。系统自动在线批改作业，显示作业成绩，并将该成绩记录在作业成绩统计文件中。

【问题 1】

如果将数据库服务器(记为 DB)作为一个外部实体，那么在绘制该系统的数据流图时，还应有哪些外部实体和数据存储？

【问题 2】

根据说明，结合问题 1 的解答，指出在该系统的顶层数据流图中应有哪些数据流？请采用说明中的词汇给出这些数据流的起点、终点及数据流名称。表 1-12 给出了数据流的部分信息，请填充空缺处。

表 1-12　数据流的部分信息

序　号	起　点	终　点	数据流名称
1	(1)	网上作业提交与管理系统	作业申请
2	(2)	网上作业提交与管理系统	提交的作业
3	网上作业提交与管理系统	(3)	需完成的作业
4	网上作业提交与管理系统	(4)	(5)
5	网上作业提交与管理系统	(6)	作业申请
6	网上作业提交与管理系统	(7)	(8)
7	(9)	网上作业提交与管理系统	选课学生名单
8	(10)	网上作业提交与管理系统	(11)
9	(12)	网上作业提交与管理系统	账号和密码
10	(13)	网上作业提交与管理系统	账号和密码

【问题 3】

根据数据流图的设计原则，阅读图 1-44 所示的数据流图，找出其中的错误之处。

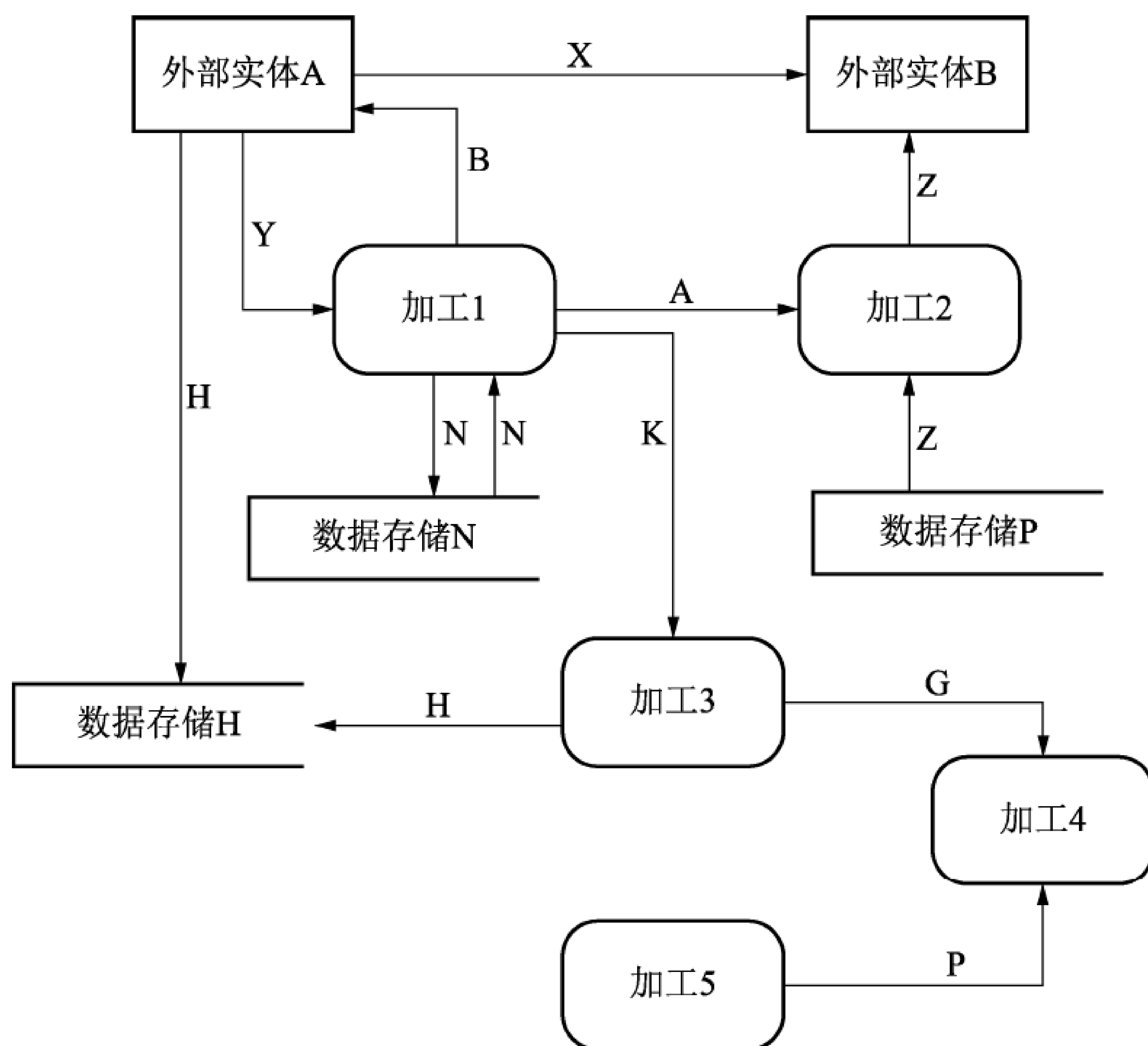


图 1-44　数据流图

1.1.4　同步练习参考答案

1.

【问题 1】

E1：商家。E2：支付系统。E3：物流系统。E4：Crystal Reports。

【问题 2】

D1：订单表。D2：商品表。D3：商品分类表。D4：购物车表。

【问题 3】

图 1-18 中缺少的数据流如表 1-13 所示。

表 1-13　图 1-18 中缺少的数据流

起　点	终　点
付款	D4 或购物车表
D4 或购物车表	提交订单
顾客表	付款
D1 或订单表	生成报表

【问题 4】

转账请求=验证码+价格+账号信息

顾客订单物流查询请求=顾客标识+{订单标识}

商家订单物流查询请求=商家标识+{订单标识}

2.

【问题 1】

E1：借阅者。E2：图书管理员。E3/E4：学生数据库/职工数据库。

【问题 2】

D1：图书表。D2：借出图书表。D3：逾期未还图书表。D4：罚金表。

【问题 3】

检查借阅者身份或检查借阅者 ID；检查逾期未还图书；检查罚金是否超过限额；借阅图书；归还图书。

【问题 4】

保持父图与子图平衡。父图中某加工的输入/输出数据流必须与其子图的输入/输出数据流在数量和名字上相同。如果父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流，而子图中组成这些数据流的数据项全体正好是父图中的这一数据流，那么它们仍然算是平衡的。

3.

【问题 1】

E1：应聘者。E2：部门经理。E3：工资系统。

【问题 2】

D1：未录用的应聘者表。D2：评价结果表。

【问题 3】

P1：验证信息。P2：审查申请。P3：职位安排评价。

【问题 4】

不平衡。图 1-22 中加工的输入/输出数据流与其子图 1-23 中的输入/输出数据流的数量不同。图 1-23 中缺少的数据流如表 1-14 所示。

表 1-14　图 1-23 中缺少的数据流

数据流名称	起　点
录用职位	P3 或 2.3(职位安排评价)
已受理的申请	1.2(受理申请)
谢绝决策	2.2(谢绝应聘者)

4.

【问题 1】

E1：病人。E2：护理人员。E3：医生。

【问题 2】

D1：生命体征范围文件。D2：日志文件。D3：病历文件。D4：治疗意见文件。

【问题 3】

图 1-25 中缺少的数据流如表 1-15 所示。

表 1-15　图 1-25 中缺少的数据流

数据流名称	起　点	终　点
重要生命体征	本地监控	格式化生命体征
格式化后的生命体征	格式化生命体征	检查生命体征
—	生成病历	D3 或病历文件
—	D2 或日志文件	生成病历

【问题 4】

E1 和 E3 之间不可以有数据流，因为数据流的起点和终点中必须有一个为加工(处理)。

5.

【问题 1】

E1：客户。E2：财务部门。E3：仓库。

【问题 2】

D1：客户文件。D2：商品文件。D3：订单文件。

【问题 3】

P1：产生配货单，其输入/输出数据流如表 1-16 所示。

表 1-16　P1 的输入/输出数据流

输入/输出数据流	数据流名称	起　点	终　点
输入数据流	订单记录	D3	P1
输出数据流	备货单	P1	E3

P2：准备发货单，其输入/输出数据流如表 1-17 所示。

表 1-17　P2 的输入/输出数据流

<table>
<tr><th>输入/输出数据流</th><th>数据流名称</th><th>起　点</th><th>终　点</th></tr>
<tr><td rowspan="2">输入数据流</td><td>订单记录</td><td>D3</td><td>P2</td></tr>
<tr><td>客户记录</td><td>D1</td><td>P2</td></tr>
<tr><td>输出数据流</td><td>发货单</td><td>P2</td><td>发货</td></tr>
</table>

图 1-27 中缺少的数据流如表 1-18 所示。

表 1-18　图 1-27 中缺少的数据流

起　点	终　点
D1	创建客户账单

6.

【问题 1】

E1：前端应用。E2：数据管理员。E3：后端数据库。

【问题 2】

D1：用户表。D2：操作表。D3：权限表。

【问题 3】

P：操作结果处理，其输入/输出数据流如表 1-19 所示。

表 1-19　P 的输入/输出数据流

输入/输出数据流	数据流名称	起　点	终　点
输入数据流	操作结果	E3	P
输出数据流	处理后的操作结果	P	E1

图 1-29 中缺少的数据流如表 1-20 所示。

表 1-20　图 1-29 中缺少的数据流

起　点	终　点
D2	权限验证
D3	权限验证

【问题 4】

在绘制数据流图的加工时，可能出现的输入、输出错误如下。

(1) 只有输入而无输出或者黑洞。

(2) 只有输出而无输入或者奇迹。

(3) 输入数据流无法通过加工产生输出数据流或者灰洞。

(4) 输入数据流与输出数据流名称相同。

7.

【问题 1】

E1：非信用卡客户。E2：信用卡客户。E3：银行。

【问题 2】

图 1-30 中缺少的 3 条数据流如表 1-21 所示。

表 1-21　图 1-30 中缺少的 3 条数据流

起　点	终　点
E1	P0 信用卡管理系统(CCMS)
P0 信用卡管理系统(CCMS)	E2
E2	P0 信用卡管理系统(CCMS)

【问题 3】

图 1-31 中错误的数据流和改正后的数据流分别如表 1-22 和表 1-23 所示。

表 1-22　图 1-31 中错误的数据流

起　点	终　点
P4	E1
P4	P3

表 1-23　图 1-31 中改正后的数据流

起　点	终　点
E1	P4
E2	P3

【问题 4】

P1：交易信息查询。P2：信用卡客户信息管理。

P3：信用卡激活。P4：信用卡申请。

8.

【问题 1】

E1：财务部门。E2：采购部门。E3：连锁超市。E4：配送部门。

D1：采购清单记录文件。D2：商品库存记录文件。D3：商品配送记录文件。D4：供货请求记录文件。

【问题 2】

图 1-32 中错误的数据流如表 1-24 所示。

表 1-24　图 1-32 中错误的数据流

起　点	终　点
D4	供货请求处理
供货请求处理	配送处理
D2	采购处理
E1	E2

图 1-32 中应补充的数据流如表 1-25 所示。

表 1-25　图 1-32 中应补充的数据流

起　点	终　点
供货请求处理	D4
供货请求处理	E4
采购处理	D2

9.

【问题 1】

E1：客户。E2：采购部门。E3：库房。E4：经理。

【问题 2】

D1：缺货记录文件。D2：库存记录文件。D3：订单记录文件。

【问题 3】

图 1-34 中缺少的数据流如表 1-26 所示。

表 1-26　图 1-34 中缺少的数据流

起　点	终　点
缺货记录文件或 D1	进货处理
订单记录文件或 D3	销售统计
库存记录文件或 D2	处理订单
进货处理	供货处理

10.

【问题 1】

E1：客户。

【问题 2】

D1：客户信息文件。D2：音像制品信息文件。D3：租借记录文件。D4：预约记录文件。

【问题 3】

图 1-36 中缺少的数据流如表 1-27 所示。

表 1-27　图 1-36 中缺少的数据流

起　点	终　点
创建预约记录	客户或 E1
归还音像制品	履行预约服务
客户或 E1	创建新客户

【问题 4】

面向数据结构的设计方法的基本思想是以数据结构作为设计的基础，它根据输入/输出数据结构导出程序的结构，适用于规模不大的数据处理系统。

11.

【问题 1】

E1：考试委员会。E2：主讲教师。E3：学生或选课学生。E4：教务处。

【问题 2】

D1：学生信息文件。D2：课程单元信息文件。D3：课程信息文件。D4：课程成绩文件。D5：无效成绩文件。

注：D2 和 D3 的答案可以互换。

【问题 3】

图 1-38 中缺少的数据流如表 1-28 所示。

表 1-28　图 1-38 中缺少的数据流

起　点	终　点
D4 或课程成绩文件	4 或生成成绩列表
D1 或学生信息文件	5 或生成最终成绩单
4 或生成成绩列表	5 或生成最终成绩单

【问题 4】

程序流程图通常在进行详细设计时使用，用来描述程序的逻辑结构。

12.

【问题 1】

(1) 费用单。(2) 待租赁房屋列表。(3) 看房请求。(4) 变更房屋状态请求。

【问题 2】

(5) 房主信息文件。(6) 租赁者信息文件。(7) 房屋信息文件。(8) 看房记录文件。

【问题 3】

(1) 起点：房主。终点：变更房屋状态。数据流名称：变更房屋状态请求。

(2) 起点：租赁者。终点：登记租赁者信息。数据流名称：租赁者信息。

(3) 起点：租赁者。终点：安排租赁者看房。数据流名称：看房请求。

13.

【问题 1】

初录数据、复录数据。

【问题 2】

0 层数据流图中，数据清除处理(加工 6)没有输入数据流。

【问题 3】

①

【问题 4】

①、②、④

【问题 5】

手工分户账=初录分户账+复录分户账

14.

【问题 1】

外部实体：(选课)学生、(任课)老师。

数据存储：作业成绩统计文件。

【问题 2】

(1) (选课)学生。(2) (选课)学生。(3) (选课)学生。(4) (选课)学生。(5) 作业成绩。(6) DB。(7) 作业成绩统计文件。(8) 作业成绩。(9) (任课)老师。(10) DB。(11) 作业。(12) (选课)学生。(13) (任课)老师。

注：(4)、(6)的答案可互换，(12)、(13)的答案可互换。

【问题 3】

错误 1：外部实体 A 和 B 之间不能存在数据流。

错误 2：外部实体 A 和数据存储 H 之间不能存在数据流。

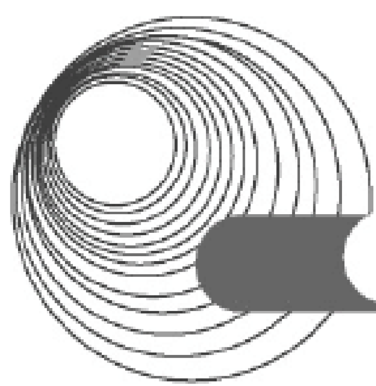
错误 3：加工 2 的输入/输出数据流名字相同。

注：若回答数据存储 P 和加工 2 的数据流方向相反也正确。

错误 4：加工 4 只有输入没有输出。

注：若回答数据流 G 的方向反了(或数据流 P 的方向反了)也正确。

错误 5：加工 5 只有输出没有输入。



1.2 本 章 小 结

本章知识点在 2009 年的新大纲中改动不大。

根据近几年软件设计师水平考试试题分布情况来看，数据流图的设计已经成为下午部分的必考题，占的分数也是一定的，几个小问题，一共 15 分。

数据流图本身的特点使得考查的题型比较集中，常出考题有：找出遗漏的数据流；指出错误/多余的数据流；找出数据流图中的多余文件等。近几年把数据字典、数据库、面向对象设计等知识结合到了数据流图中考查，但一般难度都不大。

第 2 章　数据库设计

大纲要求：

- 理解和掌握数据库管理系统的功能和特征。
- 了解数据库模型，包括概念模式、外模式和内模式。
- 了解数据模型，包括 E-R 图、第一范式、第二范式、第三范式。
- 了解数据操作，包括集合运算和关系运算。
- 了解数据库语言，即 SQL。
- 了解数据库的控制功能，包括并发控制、恢复、安全性、完整性。
- 了解数据仓库和分布式数据库的基础知识。
- 了解数据库的逻辑设计和物理设计。

2.1　数据库设计的基础知识

2.1.1　考点辅导

随着数据库系统的发展，文件系统已经慢慢被淘汰，所以在近几年的软件设计师考试中，对数据库系统的考查正逐渐取代对文件系统的考查。

数据库设计的考点和上午考题的考点重叠，只是考查方式不同。考生可以参考同系列的《软件设计师考试同步辅导(上午科目)》中的“数据库技术基础”部分。常考的考点有数据模型、主键和超键、E-R 模型转换为关系模型、SQL 语句等。下面简略给出一些设计介绍。

2.1.1.1　数据库分析与设计简介

数据库设计属于系统设计的范畴。通常把使用数据库的系统统称为数据库应用系统，把对数据库应用系统的设计简称为数据库设计。数据库设计的任务是针对一个给定的应用环境，在给定的(或选择的)硬件环境和操作系统及数据库管理系统等软件环境下，创建一个性能良好的数据库模式，建立数据库及其应用系统，使之能有效地存储和管理数据，满足各类用户的需求。

合理的数据库结构是数据库应用系统性能良好的基础和保证，但数据库的设计和开发却是一项庞大而复杂的工程。从事数据库设计的人员，不仅要具备数据库知识和数据库设计技术，还要有程序开发的实际经验，掌握软件工程的原理和方法。数据库设计人员必须深入应用环境，了解用户具体的专业业务。在数据库设计的前期和后期，与应用单位人员密切联系，共同开发，可大大提高数据库设计的成功率。

2.1.1.2　数据库设计的步骤

1. 数据库应用系统的生命周期

按照软件工程对系统生命周期的定义，可把软件生命周期分为 6 个阶段：制订计划、

需求分析、设计、程序编制、测试及运行维护。在数据库设计中也参照这种划分，把数据库应用系统的生命周期分为数据库规划、需求描述与分析、数据库与应用程序设计、数据库系统实现、测试和运行维护 6 个阶段。

(1) 数据库规划。数据库规划是创建数据库应用系统的起点，是数据库应用系统的任务陈述和任务目标。任务陈述定义了数据库应用系统的主要目标，而每个任务目标定义了系统必须支持的特定任务。数据库规划过程还必然包括对工作量的估计、使用的资源和需要的经费等。同时，还应当定义系统的范围和边界，以及它与公司信息系统其他部分的接口。

(2) 需求描述与分析。需求描述与分析是站在用户的角度，从系统中的数据和业务规则入手，收集和整理用户的信息，以特定的方式加以描述，是下一步工作的基础。

(3) 数据库与应用程序设计。数据库设计是对用户数据的组织和存储设计；应用程序设计是在数据库设计的基础上对数据操作及业务实现的设计，包括事务设计和用户界面设计。

(4) 数据库系统实现。数据库系统实现是依照设计，使用 DBMS(数据库管理系统)支持的数据定义语言(DDL)实现数据库的建立，用高级语言(Basic、Delphi、C、C++和PowerBuilder 等)编写应用程序。

(5) 测试。测试是在数据系统投入使用之前，通过精心制订的测试计划和测试数据来测试系统的性能是否满足设计要求，并发现问题。

(6) 运行维护。数据库应用系统经过测试、试运行后即可正式投入运行。运行维护是系统投入使用后，必须不断地对其进行评价、调整与修改，直至系统消亡。

在任一设计阶段，一旦发现不能满足用户数据需求时，均需返回到前面的适当阶段进行必要的修正。经过如此的迭代求精过程，直到能满足用户需求为止。在进行数据库结构设计时，应考虑满足数据库中数据处理的要求，将数据和功能两方面的需求分析、设计和实现在各个阶段同时进行，相互参照和补充。

在数据库设计中，每一个阶段的设计成果都应该通过评审。评审的目的是确认某一阶段的任务是否全部完成，从而避免出现重大的错误或疏漏，保证设计质量。评审后还需要根据评审意见修改所提交的设计成果，有时甚至要回溯到前面的某一阶段，进行部分重新设计乃至全部重新设计，然后再进行评审，直至达到系统的预期目标为止。

2. 数据库设计的方法

在确定了数据库设计的策略以后，就需要应用相应的设计方法和步骤。多年来，人们提出了多种数据库设计方法、设计准则和规范。

1978 年 10 月召开的新奥尔良(New Orleans)会议提出的关于数据库设计的步骤(简称新奥尔良法)是目前得到公认的、较完整的、较权威的数据库设计方法，它把数据库设计分为以下 4 个主要阶段。

(1) 用户需求分析。用户需求分析是数据库设计人员采用一定的辅助工具对应用对象的功能、性能和限制等要求所进行的科学分析。

(2) 概念设计。概念设计是对信息进行分析和定义，如视图模型化、视图分析和汇总。该阶段对应用对象精确地进行抽象和概括，以形成独立于计算机系统的企业信息模型。描述概念模型的较理想工具是 E-R 图。

(3) 逻辑设计。逻辑设计是将抽象的概念模型转化为与选用的 DBMS 产品所支持的数据模型相符合的逻辑模型，它是物理设计的基础，包括模式初始设计、子模式设计、应用

程序设计、模式评价及模式求精。

(4) 物理设计。物理设计是将逻辑模型转化为计算机中的具体实现方案。

在任一阶段，当发现不能满足用户需求时，均需返回到前面的适当阶段进行必要的修正。经过不断的迭代求精，直到各种性能均能满足用户的需求为止。

2.1.2　典型例题分析

例 1　某销售公司当前的销售业务为商城实体店销售。现该公司拟开展网络销售业务，需要开发一个信息化管理系统。请根据公司现有业务及需求完成该系统的数据库设计。(2016 年 5 月试题二)

【需求描述】

(1) 记录公司所有员工的信息。员工信息包括工号、身份证号、姓名、性别、出生日期和电话，并只登记一部电话。

(2) 记录所有商品的信息。商品信息包括商品名称、生产厂家、销售价格和商品介绍。系统内部用商品条码唯一区别每种商品。

(3) 记录所有顾客的信息。顾客信息包括顾客姓名、身份证号、登录名、登录密码和电话号码。一位顾客只能提供一个电话号码。系统自动生成唯一的顾客编号。

(4) 顾客登录系统之后，在网上商城购买商品。顾客可将选购的商品置入虚拟的购物车内，购物车可长期存放顾客选购的所有商品。顾客可在购物车内选择商品、修改商品数量后生成网购订单。订单生成后，由顾客选择系统提供的备选第三方支付平台进行电子支付，支付成功后系统需要记录唯一的支付凭证编号，然后由商城根据订单进行线下配送。

(5) 所有的配送商品均由仓库统一出库。为方便顾客，允许每位顾客在系统中提供多组收货地址、收货人及联系电话。一份订单所含的多个商品可能由多名分拣员根据商品所在仓库信息从仓库中进行分拣操作，分拣后的商品交由配送员根据配送单上的收货地址进行配送。

(6) 新设计的系统要求记录实体店的每笔销售信息，包括营业员、顾客、所售商品及其数量。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-1 所示。

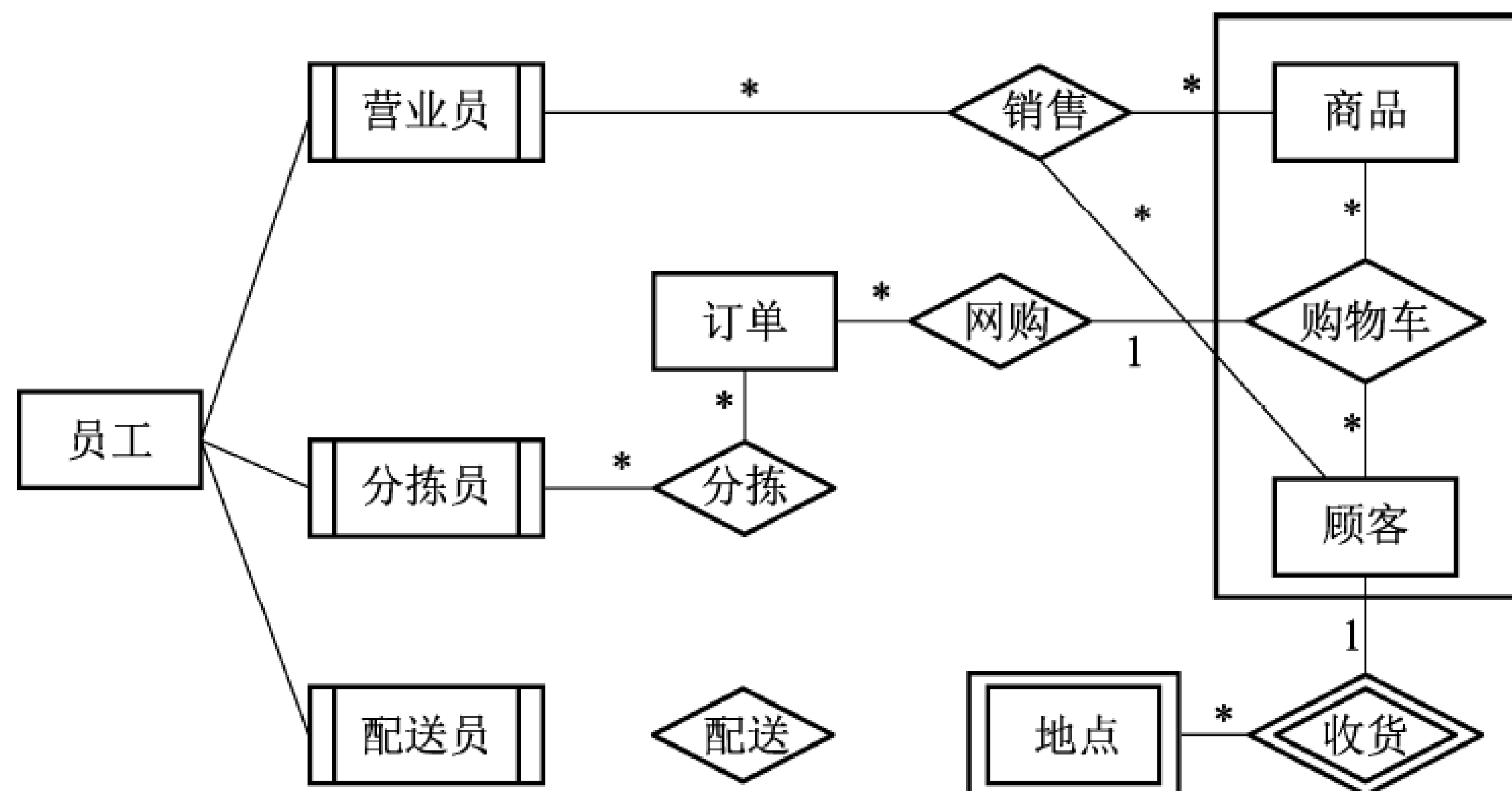


图 2-1　例 1 实体联系图(不完整)

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

员工(工号，身份证号，姓名，性别，出生日期，电话)；

商品(商品条码，商品名称，生产厂家，销售价格，商品介绍，(a))；

顾客(顾客编号，姓名，身份证号，登录名，登录密码，电话)；

收货地点(收货 ID，顾客编号，收货地址，收货人，联系电话)；

购物车(顾客编号，商品条码，商品数量)；

订单(订单 ID，顾客编号，商品条码，商品数量，(b))；

分拣(分拣 ID，分拣员工号，(c)，分拣时间)；

配送(配送 ID，分拣 ID，配送员工号，收货 ID，配送时间，签收时间，签收快照)；

销售(销售 ID，营业员工号，顾客编号，商品条码，商品数量)。

【问题 1】(4 分)

补充图 2-1 中的“配送”联系所关联的对象及联系类型。

【问题 2】(6 分)

补充逻辑结构设计中的(a)、(b)和(c)三处空缺。

【问题 3】(5 分)

对于实体店销售，若要增加送货上门服务，由营业员在系统中下订单，与网购的订单进行后续的统一管理。请根据该需求，对图 2-1 进行补充，并修改订单关系模式。

解析：

本题考查数据库系统中实体联系模型和关系模式设计方面的应用知识，属于比较传统的题目，考查点和往年类似。

【问题 1】

本题考查数据库的概念结构设计，两个实体集间的联系分为三类：一对一(1∶1)、一对多(1∶n)和多对多(m∶n)。

根据题意，一名配送员可以配送到多个地点，一个地点也可有多名配送员配送，所以配送员与地点之间是多对多的关系。

【问题 2】

本题考查数据库的逻辑结构设计，题目要求补充完整各关系模式。根据需求描述，商品关系模式除了应包括商品名称、生产厂家、销售价格和用于区别商品的唯一商品条码，还应有数量说明，另外，所有商品配送均由仓库统一出库，所以应有商品的仓库信息，即(a)处应填商品数量和仓库信息；根据需求描述(4)可知，订单关系模式还缺少支付凭证编号，(b)处应填入支付凭证编号；在分拣关系模式中除了包含分拣 ID、分拣员工号、分拣时间，还应该有分拣的是哪种商品及该商品的订单信息，所以(c)处应填商品条码、订单 ID。

【问题 3】

对于实体店销售，若要增加送货上门服务，可参照顾客关系模式，增加营业员关系模式，在订单关系模式中增加“营业员 ID”属性，并设为外键，实现与营业员关系模式联系。

答案：

【问题 1】

配送员—配送—地点，多对多联系。

【问题 2】

(a) 商品数量，仓库信息;

(b) 支付凭证编号;

(c) 商品条码，订单 ID。

【问题 3】

在营业员与订单之间增加联系“送货上门”，并将联系记录于订单关系中，即在订单关系中增加属性“营业员 ID”作为外键。

例 2　某企业拟构建一个高效、低成本、符合企业实际发展需要的办公自动化系统。工程师小李主要承担该系统的公告管理和消息管理模块的研发工作。公告管理模块的主要功能包括添加、修改、删除和查看公告。消息管理模块的主要功能是消息群发。(2015 年 11 月试题二)

小李根据前期调研和需求分析进行了概念模型设计，具体情况分述如下。

【需求分析结果】

(1) 该企业设有研发部、财务部、销售部等多个部门，每个部门只有一名部门经理，有多名员工，每名员工只属于一个部门，部门信息包括：部门号、名称、部门经理和电话，其中部门号唯一确定部门关系的每一个元组。

(2) 员工信息包括：员工号、姓名、岗位、电话和密码。员工号唯一确定员工关系的每一个元组；岗位主要有经理、部门经理、管理员等，不同岗位具有不同的权限。一名员工只对应一个岗位，但一个岗位可对应多名员工。

(3) 消息信息包括：编号、内容、消息类型、接收人、接收时间、发送时间和发送人。其中(编号，接收人)唯一标识消息关系中的每一个元组。一条消息可以发送给多个接收人，一个接收人可以接收多条消息。

(4) 公告信息包括：编号、标题、名称、内容、发布部门、发布时间。其中编号唯一确定公告关系的每一个元组。一份公告对应一个发布部门，但一个部门可以发布多份公告；一份公告可以被多名员工阅读，一名员工可以阅读多份公告。

【概念模型设计】

根据需求分析阶段收集的信息，设计的实体联系图(不完整)如图 2-2 所示。

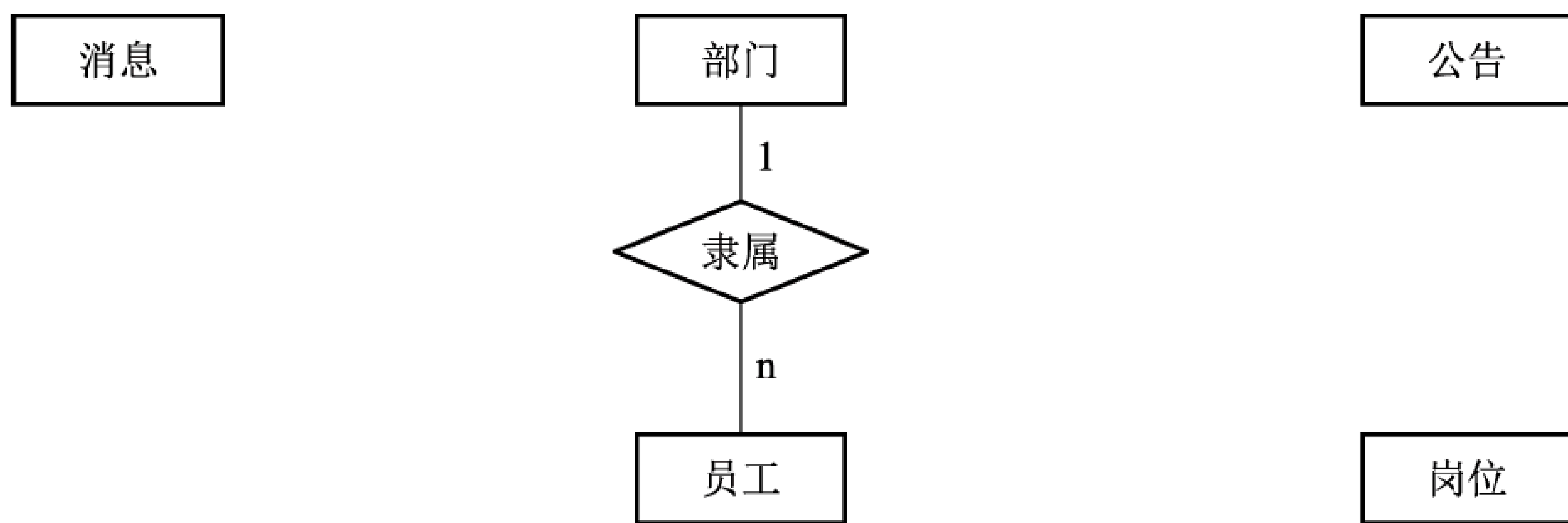


图 2-2　例 2 实体联系图(不完整)

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整):

部门((a) ，部门经理，电话);
员工(员工号，姓名，岗位号，部门号，电话，密码);
岗位(岗位号，名称，权限);
消息((b) ，消息类型，接收时间，发送时间，发送人);
公告((c) ，名称，内容，发布部门，发布时间);
阅读公告((d) ，阅读时间)。

【问题 1】(5 分)

根据问题描述，补充四个联系，完善图 2-2 所示的实体联系图。联系名可用联系 1、联系 2、联系 3 和联系 4 代替，联系的类型分为 1∶1、1∶n 和 m∶n(或 1∶1、1∶*和*∶*)。

【问题 2】(8 分)

(1) 根据实体联系图，将关系模式中的空(a)~(d)补充完整。

(2) 给出“消息”和“阅读公告”关系模式的主键与外键。

【问题 3】(2 分)

消息和公告关系中都有“编号”属性，请问它是属于命名冲突吗？用 100 字以内文字说明原因。

解析：

【问题 1】

每个部门有多名员工，每名员工只属于一个部门，所以部门和员工之间是一对多的关系。一份公告可以被多名员工阅读，一名员工可以阅读多份公告，可见公告和员工之间是多对多的关系。一份公告对应一个发布部门，但一个部门可以发布多份公告，所以部门和公告之间是一对多的关系。一名员工只对应一个岗位，但一个岗位可对应多名员工，所以岗位和员工之间是一对多的关系。一条消息可以发送给多个接收人，一个接收人可以接收多条消息，所以员工和消息之间是多对多的关系。

【问题 2】

部门信息包括：部门号、名称、部门经理和电话，部门关系模式应有部门号、名称、部门经理和电话 4 个属性，部门号是主码；消息信息包括：编号、内容、接收人、消息类型、接收时间、发送时间和发送人，消息关系模式应有编号、内容、接收人、消息类型、接收时间、发送时间和发送人 7 个属性，(编号，接收人)是主码；公告信息包括：编号、标题、名称、内容、发布部门、发布时间，公告关系模式应有编号、标题、名称、内容、发布部门、发布时间 6 个属性；一份公告可以被多名员工阅读，公告编号唯一确定公告关系的每一个元组，所以阅读公告关系模式应由员工号、公告编号、阅读时间组成，(员工号，公告编号)是主码。

【问题 3】略。

答案：

【问题 1】

完善的实体联系图如图 2-3 所示。

【问题 2】

(a) 部门号，名称；

(b) 编号，内容，接收人；

(c) 编号，标题；

(d) 员工号，公告编号。

消息的主键：(编号，接收人)；外键：接收人，发送人。

阅读公告的主键：(员工号，公告编号)；外键：员工号，公告编号。

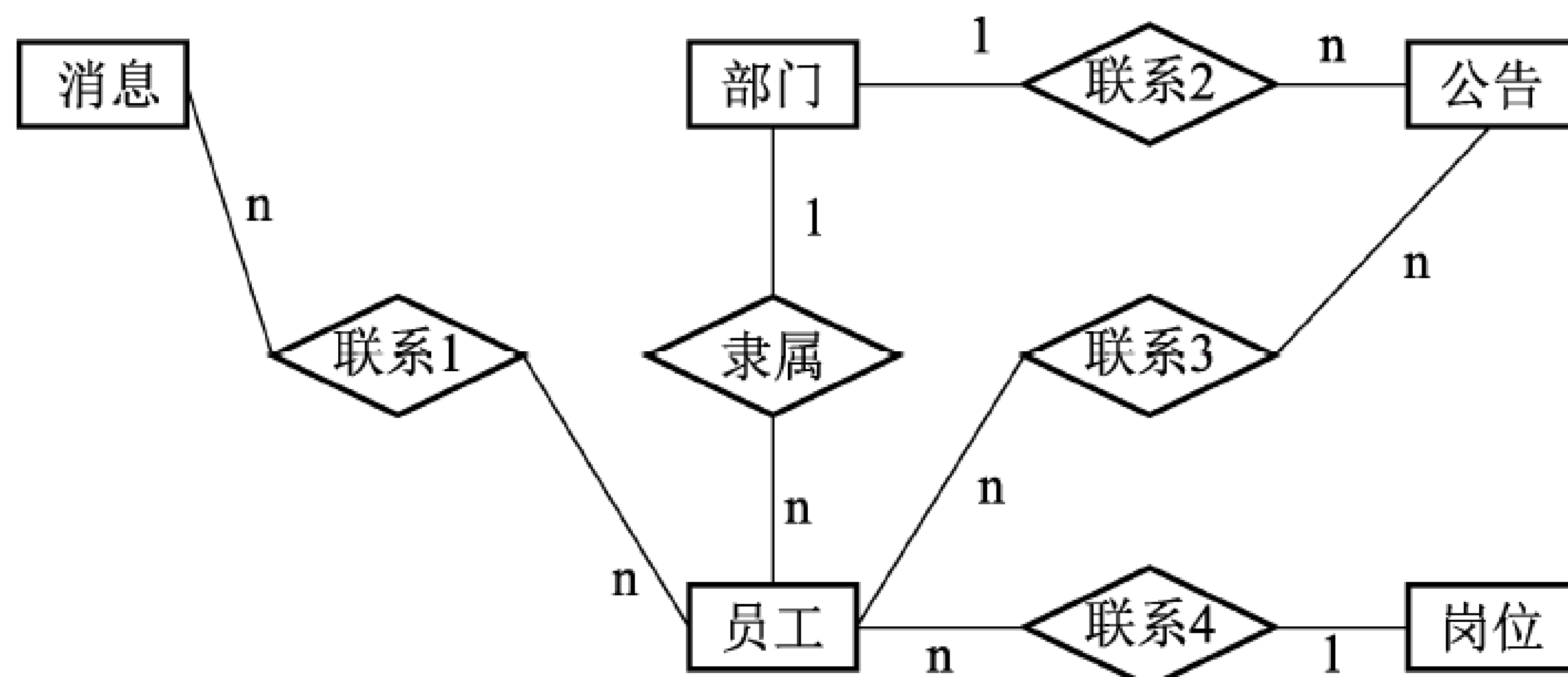


图 2-3　例 2 完善的实体联系图

【问题 3】

不属于命名冲突。

命名冲突是在合并 E-R 模型时提出的概念，合并 E-R 模型时之所以产生冲突，是因为对于同样的对象，不同的局部 E-R 模型有着不同的定义，在本题中，本就是不同对象的属性，所以不存在冲突的说法。

例 3　某省针对每年举行的足球联赛，拟开发一套信息管理系统，以方便管理球队、球员、主教练、主裁判、比赛等信息。(2015 年 5 月试题二)

【需求分析】

(1) 系统需要维护球队、球员、主教练、主裁判、比赛等信息。

球队信息主要包括：球队编号、名称、成立时间、人数、主场地址、球队主教练。

球员信息主要包括：姓名、身份证号、出生日期、身高、家庭住址。

主教练信息主要包括：姓名、身份证号、出生日期、资格证书号、级别。

主裁判信息主要包括：姓名、身份证号、出生日期、资格证书号、获取证书时间、级别。

(2) 每支球队有一名主教练和若干名球员。一名主教练只能受聘于一支球队，一名球员只能效力于一支球队。每支球队都有自己的唯一主场场地，且场地不能共用。

(3) 足球联赛采用主客场循环制，一周进行一轮比赛，一轮的所有比赛同时进行。

(4) 一场比赛有两支球队参加，一支球队作为主队身份、另一支作为客队身份参与比赛。一场比赛只能有一名主裁判，每场比赛有唯一的比赛编码，每场比赛都记录比分和日期。

【概念结构设计】

根据需求分析阶段的信息，设计的实体联系图(不完整)如图 2-4 所示。

【逻辑结构设计】

根据概念结构设计阶段完成的实体联系图，得出如下关系模式(不完整)：

球队(球队编号，名称，成立时间，人数，主场地址)；

球员(姓名，身份证号，出生日期，身高，家庭住址，　(1))；

主教练(姓名，身份证号，出生日期，资格证书号，级别， (2))；
主裁判(姓名，身份证号，出生日期，资格证书号，获取证书时间，级别)；
比赛(比赛编码，主队编号，客队编号，主裁判身份证号，比分，日期)。

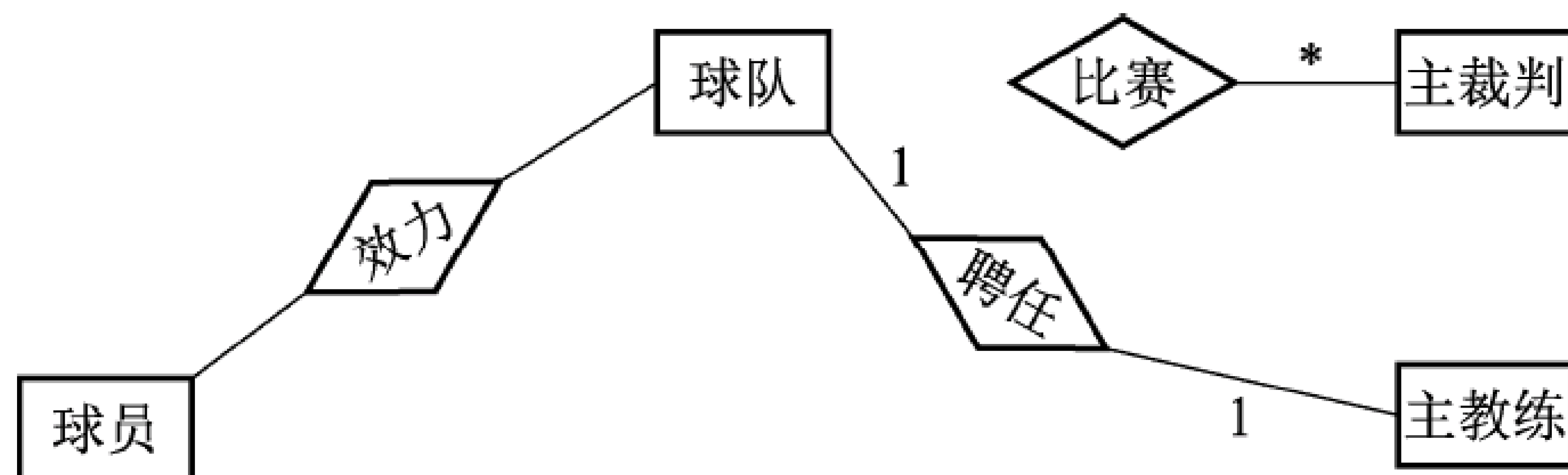


图 2-4 例 3 实体联系图(不完整)

【问题 1】(6 分)

补充图 2-4 中的联系和联系的类型。

图 2-4 中的联系“比赛”应具有的属性是哪些?

【问题 2】(4 分)

根据图 2-4，将逻辑结构设计阶段生成的关系模式中的空(1)、(2)补充完整。

【问题 3】(5 分)

现在系统要增加赞助商信息，赞助商信息主要包括赞助商名称和赞助商编号。

赞助商可以赞助某支球队，一支球队只能有一个赞助商，但赞助商可以赞助多支球队。赞助商也可以单独赞助某些球员，一名球员可以为多个赞助商代言。请根据该要求，对图 2-4 进行修改，画出修改后的实体间联系和联系的类型。

解析：

本题考查数据库设计，涉及的考点有数据库概念结构设计和逻辑结构设计。

【问题 1】

联系的类型分为 1∶1、1∶n 和 m∶n(或 1∶1、1∶*和*∶*)，该题中球队与主裁判只有通过比赛联系在一起，联系类型为*∶*；根据题目需求分析可知，每场比赛有唯一的比赛编码，且每场比赛都记录比分和日期，所以图 2-4 中的联系“比赛”应具有比赛编码、比分和日期三个属性。

【问题 2】

根据题目需求分析可知球员信息主要包括：姓名、身份证号、出生日期、身高、家庭住址。但球员属于球队，比赛以球队为单位组织进行，所以球员关系模式中缺失球队编号，即(1)处应填球队编号；同理，也不难分析出主教练关系模式中也缺少球队编号，即(2)处填入球队编号。

【问题 3】

根据题目中问题 3 描述，增加的实体赞助商与球队实体和球员实体存在赞助联系，根据问题描述很容易得到赞助商与球队之间的联系类型为 1∶*，与球员的联系类型为*∶*。

答案：

【问题 1】

完善的实体联系图如图 2-5 所示。

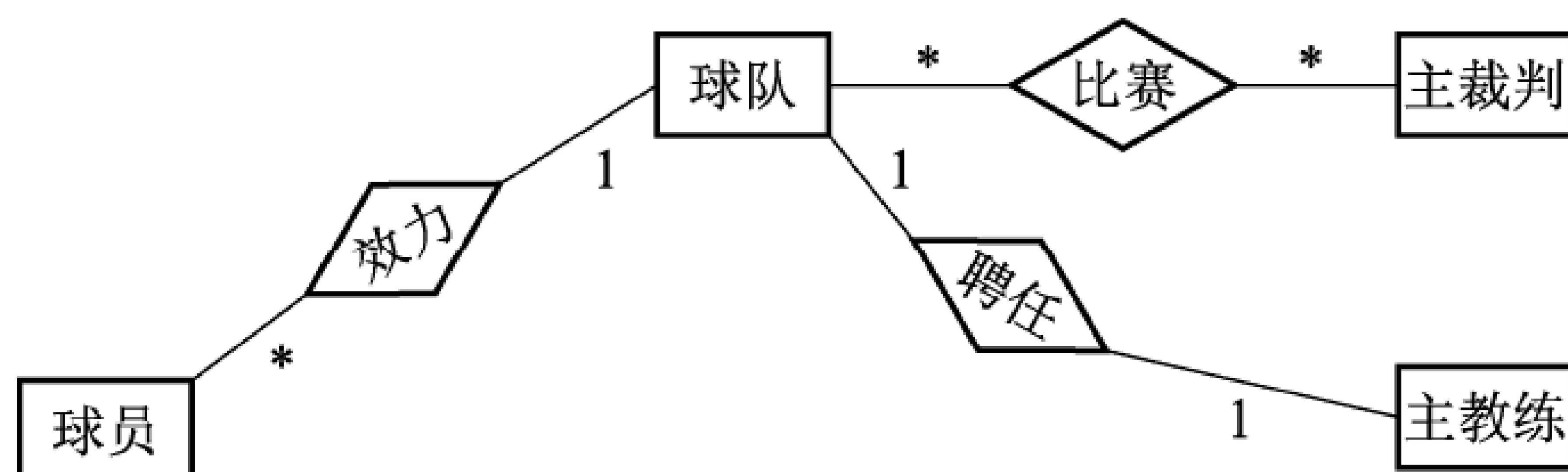


图 2-5　例 3 完善的实体联系图

比赛联系应具有的属性包括：比赛编码，比分，日期。

【问题 2】

(1)球队编号；(2)球队编号。

【问题 3】

修改后的实体联系图如图 2-6 所示。

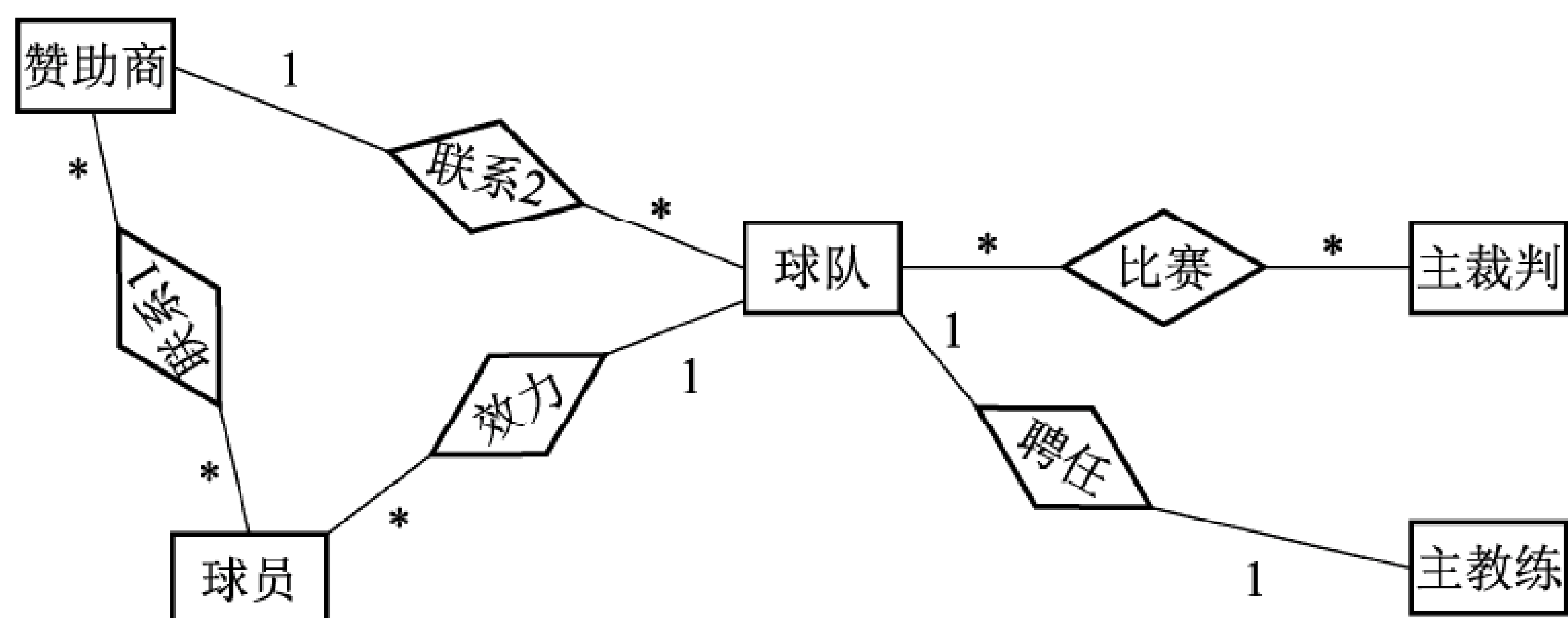


图 2-6　例 3 修改后的实体联系图

例 4　某集团公司在全国不同城市拥有多个大型超市，为了有效管理各个超市的业务工作，需要构建一个超市信息管理系统。(2014 年 11 月试题二)

【需求分析结果】

(1) 超市信息包括：超市名称、地址、经理和电话，其中超市名称唯一确定超市关系的每一个元组。每个超市只有一名经理。

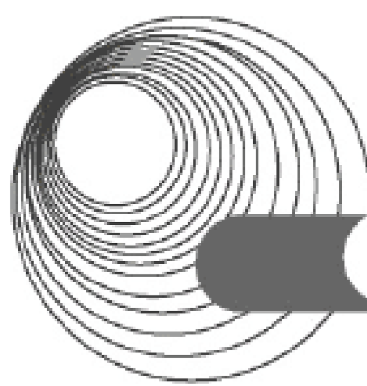
(2) 超市设有计划部、财务部、销售部等多个部门，每个部门只有一名部门经理，有多名员工，每个员工只属于一个部门。部门信息包括：超市名称、部门名称、部门经理和联系电话。超市名称、部门名称唯一确定部门关系的每一个元组。

(3) 员工信息包括：员工号、姓名、超市名称、部门名称、职位、联系方式和工资。其中，职位信息包括：经理、部门经理、业务员等。员工号唯一确定员工关系的每一个元组。

(4) 商品信息包括：商品号、商品名称、型号、单价和数量。商品号唯一确定商品关系的每一个元组。一名业务员可以负责超市内多种商品的配给，一种商品可以由多名业务员配给。

【概念模型设计】

根据需求分析阶段收集的信息，设计的实体联系图和关系模式(不完整)如图 2-7 所示。



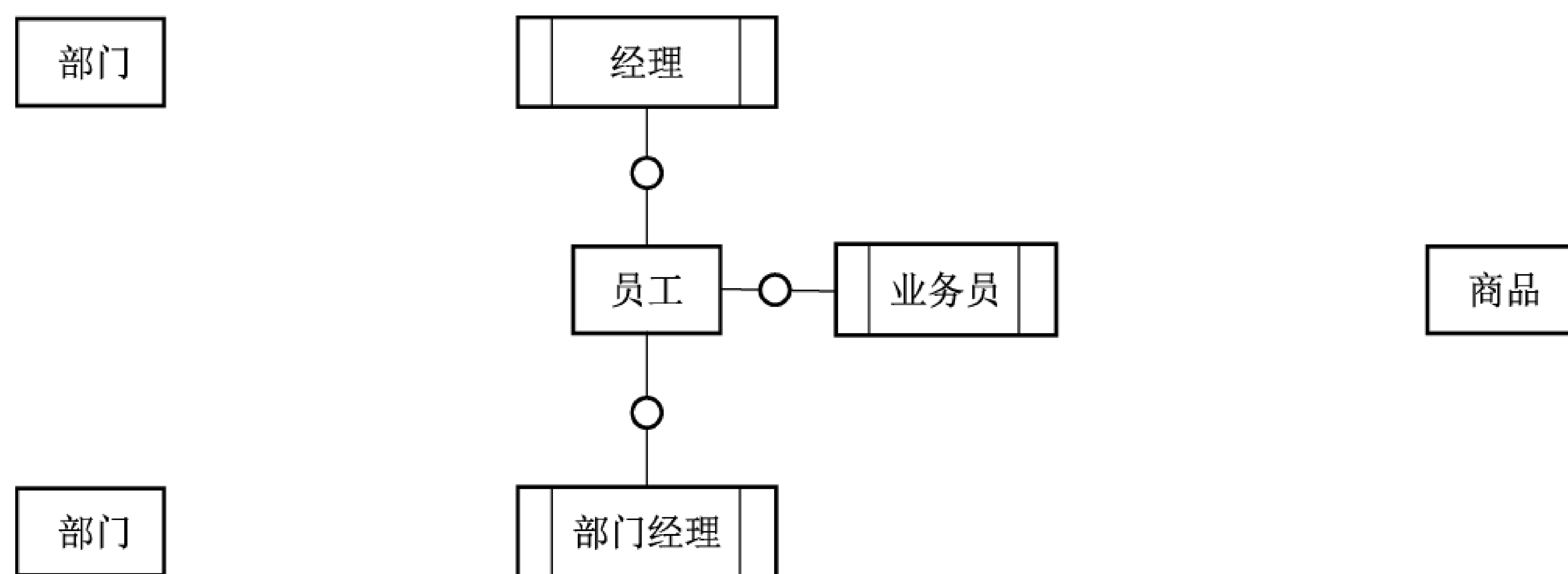


图 2-7　例 4 实体联系图(不完整)

【关系模式设计】

超市(超市名称，经理，地址，电话)；

部门((a)，部门经理，联系电话)；

员工((b)，姓名，联系方式，职位，工资)；

商品(商品号，商品名称，型号，单价，数量)；

配给((c)，配给时间，配给数量，业务员)。

【问题 1】(4 分)

根据问题描述，补充四个联系，完善图 2-7 的实体联系图。联系名可用联系 1、联系 2、联系 3 和联系 4 代替，联系的类型分为 1∶1、1∶n 和 m∶n(或 1∶1、1∶*和*∶*)。

【问题 2】(7 分)

(1) 根据实体联系图，将关系模式中的空(a)~(c)补充完整。

(2) 给出部门和配给关系模式的主键和外键。

【问题 3】(4 分)

(1) 超市关系的地址可以进一步分为邮编、省、市、街道，那么该属性是属于简单属性还是复合属性？请用 100 字以内文字说明。

(2) 假设超市需要增设一个经理的职位，那么超市与经理之间的联系类型应修改为(d)，超市关系应修改为(e)。

解析：

本题考查数据库系统中实体联系模型和关系模式设计方面的应用知识，属于比较传统的题目，考查点也和往年类似。

【问题 1】

本题考查数据库的概念结构设计。两个实体集间的联系分为三类：一对一(1∶1)、一对多(1∶n)和多对多(m∶n)。

根据题意，“每个部门只有一名部门经理”，部门和部门经理之间是 1∶1 关系，每个部门有多名员工，即部门和员工之间是 1∶n 的关系，由“超市只有一名经理”可知超市的部门和经理之间是 1∶1 的关系，由“超市有多个部门”可知超市和部门之间是 1∶n 的关系。由“一名业务员可以负责超市内多种商品的配给，一种商品可以由多名业务员配给”可知业务员和商品之间是多对多的关系，即 m∶n。

根据以上分析，即可完成图 2-7 的实体联系图。

【问题 2】

本题考查数据库的逻辑结构设计，题目要求补充完整各关系模式，并给出部门和配给关系模式的主键和外键。

根据问题 1 画完整的实体联系图和需求描述，员工关系模式包括员工号、姓名、部门名称、超市名称、职位、联系方式、工资等，因此(b)处应填员工号、超市名称和部门名称；部门关系模式包括超市名称、部门名称、部门经理和联系电话等，由此可知(a)处填超市名称和部门名称。配给关系模式包括商品号、业务员、配给时间、配给数量等，由此可知，(c)处应填商品号。部门关系模式的主键应为(超市名称和部门名称)，外键名称为(超市名称、部门经理)；配给关系模式的主键应为(商品号、业务员、配给时间)，外键应为(业务员、商品号)。

【问题 3】

超市关系的地址可以进一步分为邮编、省、市、街道，则超市关系中的地址属于复合属性。所谓复合属性就是指属性中含有多种信息，可以进一步拆分的属性，地址可以拆分成多个简单属性，符合这一特征。

假设超市需要增设一个经理的职位，那么超市与经理之间的联系类型应该是一个超市有多名经理，为 1∶n 的关系，超市关系应包含超市名称、地址、电话。

答案：

【问题 1】

完善的实体联系图如图 2-8 所示。

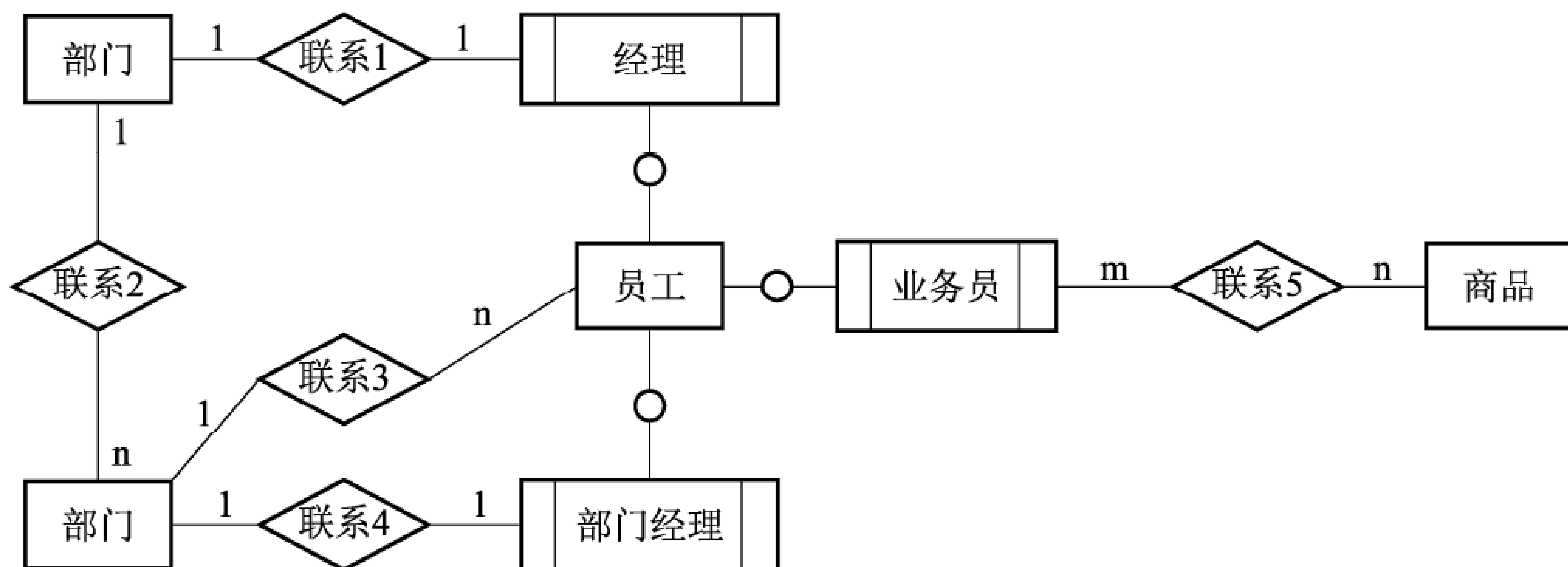


图 2-8　例 4 完善的实体联系图

【问题 2】

(a) 超市名称、部门名称。主键：(超市名称，部门名称)；外键：(超市名称、部门经理)。

(b) 员工号、超市名称、部门名称。

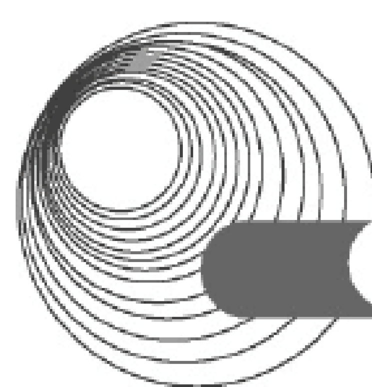
(c) 商品号。主键：(商品号，业务员，配给时间)；外键：(业务员、商品号)。

【问题 3】

(1) 超市关系中的地址属于复合属性。

(2) (d)1∶n；(e)超市名称、地址、电话。

例 5　某家电销售电子商务公司拟开发一套信息管理系统，以方便对公司的员工、家电销售、家电厂商和客户等进行管理。(2014 年 5 月试题二)



【需求分析】

(1) 系统需要维护电子商务公司的员工信息、客户信息、家电信息和家电厂商信息等。员工信息主要包括：工号、姓名、性别、岗位、身份证号、电话、住址，其中岗位包括部门经理和客服等。客户信息主要包括：客户 ID、姓名、身份证号、电话、住址、账户余额。家电信息主要包括：家电条码、家电名称、价格、出厂日期、所属商。家电厂商信息包括：厂商 ID、厂商名称、电话、法人代表信息、厂址。

(2) 电子商务公司根据销售情况，由部门经理向家电厂商订购各类家电，每个家电厂商只能由一名部门经理负责。

(3) 客户通过浏览电子商务公司网站查询家电信息，与客服沟通获得优惠后，在线购买。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-9 所示。

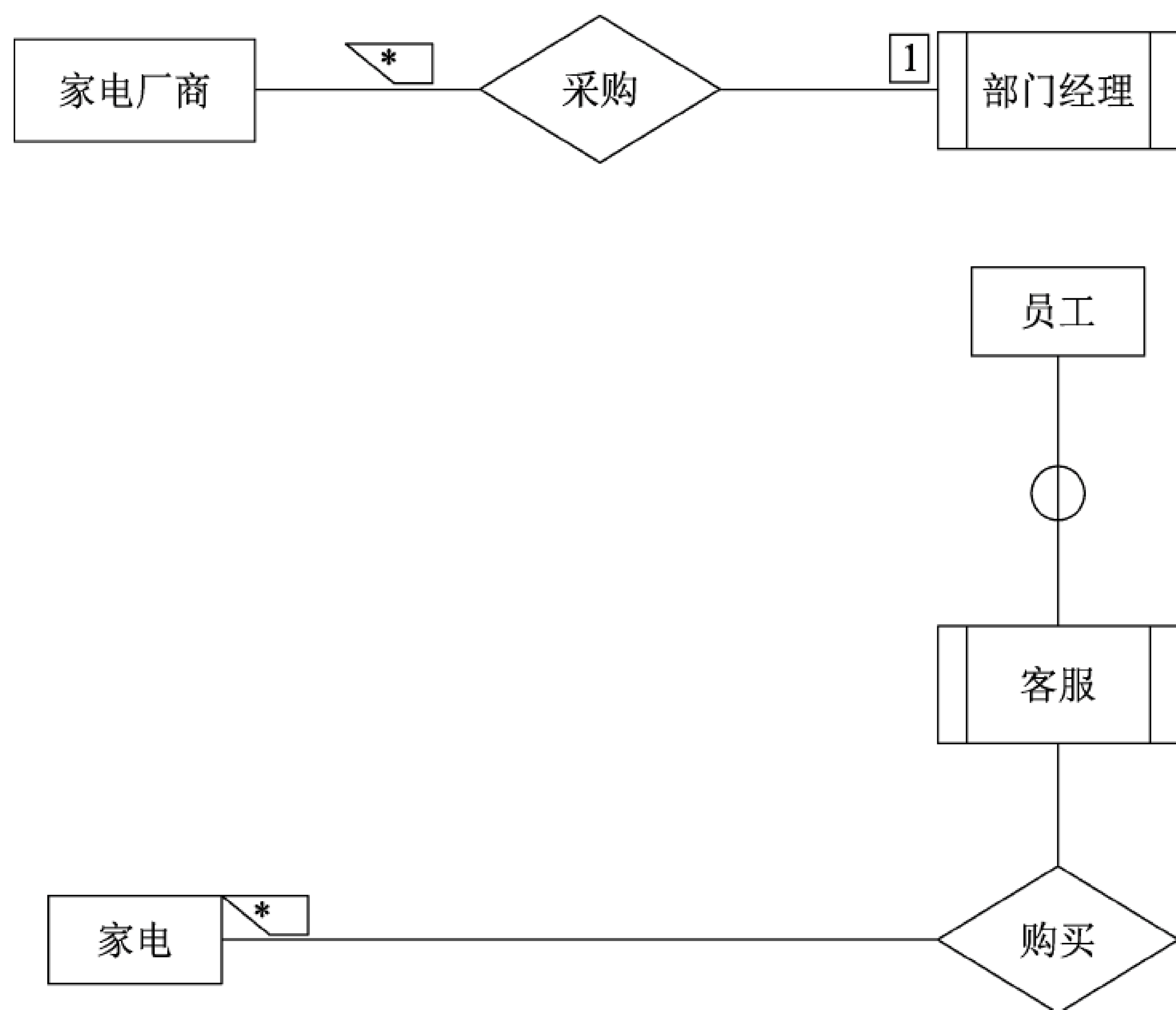


图 2-9　例 5 实体联系图(不完整)

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

客户(客户 ID，姓名，身份证号，电话，住址，账户余额)；

员工(工号，姓名，性别，岗位，身份证号，电话，住址)；

家电(家电条码，家电名称，价格，出厂日期，(1))；

家电厂商(厂商 ID，厂商名称，电话，法人代表信息，厂址，(2))；

购买(订购单号，(3)，金额)。

【问题 1】(6 分)

补充图 2-9 中的联系和联系的类型。

【问题 2】(6 分)

根据图 2-10，将逻辑结构设计阶段生成的关系模式中的空(1)~(3)补充完整，用下画线

指出“家电”“家电厂商”和“购买”关系模式的主键。

【问题 3】(3 分)

电子商务公司的主营业务是销售各类家电，对账户有余额的客户，还可以联合第二方基金公司提供理财服务，为此设立客户经理岗位。客户通过电子商务公司的客户经理和基金公司的基金经理进行理财，每名客户只有一名客户经理和一名基金经理负责，客户经理和基金经理均可负责多名客户。请根据该要求，对图 2-9 进行修改，画出修改后的实体间联系和联系的类型。

解析：

本题考查数据库设计。涉及的考点有数据库的概念结构设计和逻辑结构设计。

【问题 1】

根据题意，每个家电厂商可以购买多种家电，因此家电厂商和家电之间是一对多的联系。一种家电可由多个客户购买，一个客户可购买多件家电，因此家电和客户之间是多对多的联系，由此可画出完整的实体联系图。

【问题 2】

电子商务公司根据销售情况，由部门经理向家电厂商订购各类家电，每个家电厂商只能由一名部门经理负责，家电厂商中需要保存是由哪名部门经理负责的，因此这里需要有员工的工号，这样才能在员工中找到唯一对应的员工。

【问题 3】

由题意可知，可增设实体客户经理和基金经理，客户和客户经理、基金经理都有联系，且客户经理和客户、基金经理和客户都是一对多的联系，由此可补充图 2-9。

答案：

【问题 1】

完善的实体联系图如图 2-10 所示。

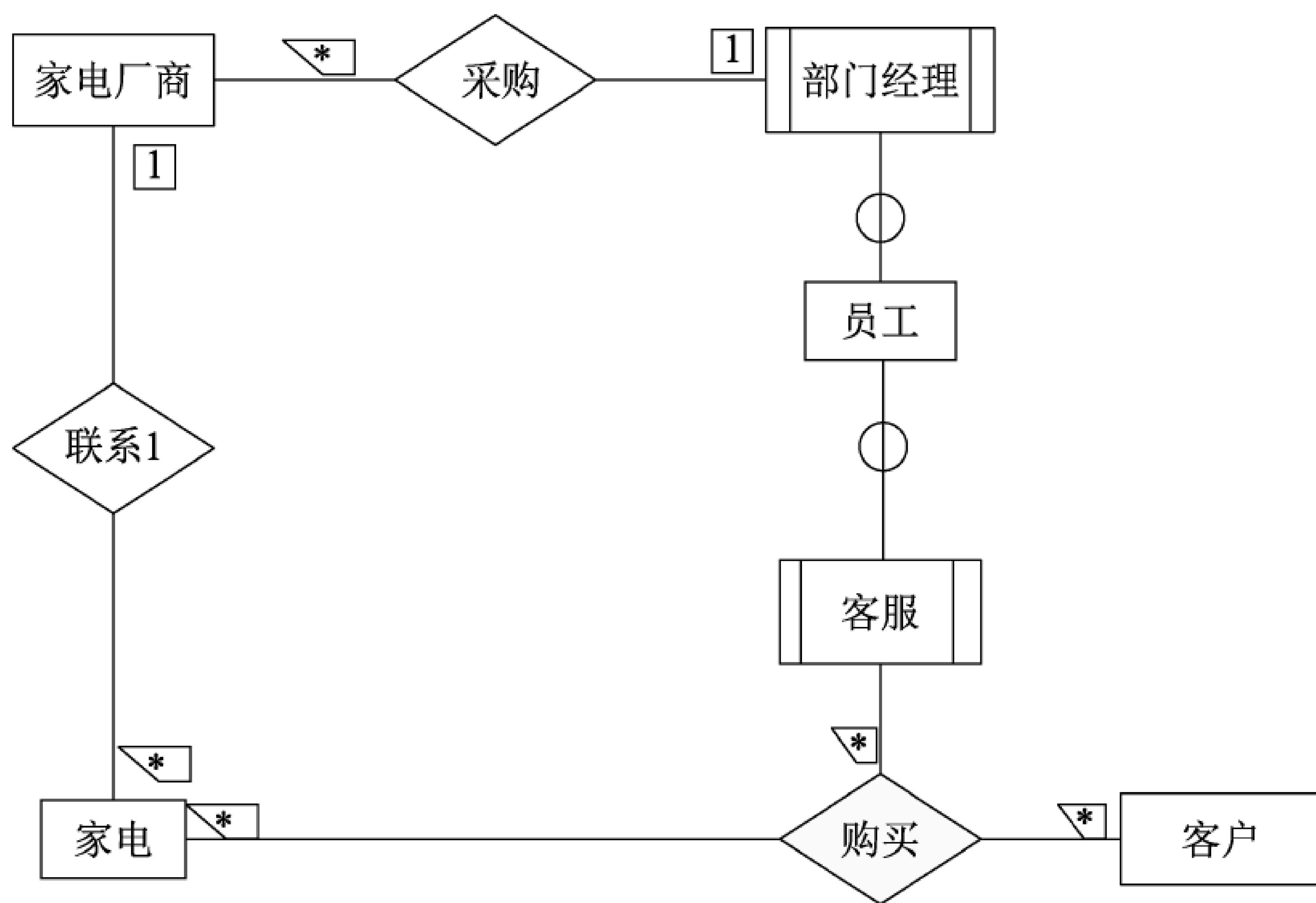
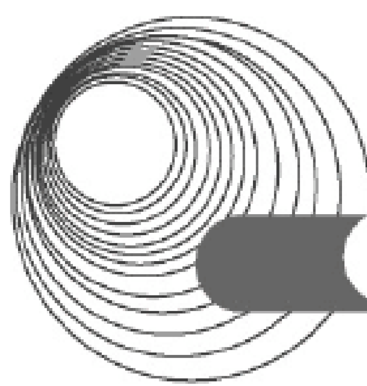


图 2-10　例 5 完善的实体联系图



【问题2】

(1) 厂商ID;

(2) 工号;

(3) 家电条码，客户ID，工号。

家电关系的主键：家电条码;

家电厂商关系的主键：厂商ID;

购买关系的主键：订购单号。

【问题3】

修改的实体联系图如图2-11所示。

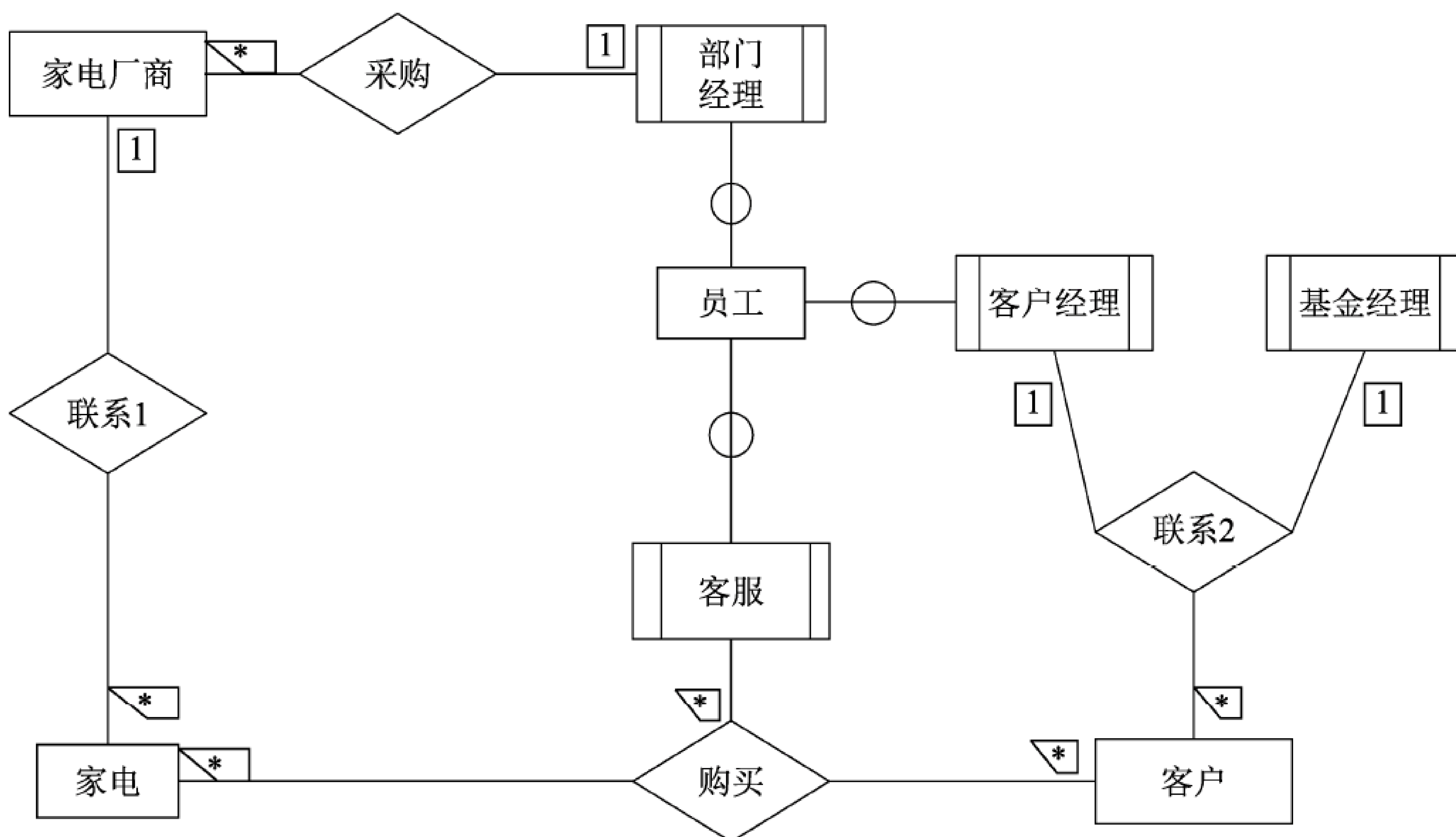


图2-11　例5修改的实体联系图

例6　某快递公司为了方便管理公司物品运送的各项业务活动，需要构建一个物品运送信息管理系统。(2013年11月试题二)

【需求分析结果】

(1) 快递公司有多个分公司，分公司信息包括分公司编号、名称、经理、办公电话和地址。每个分公司可以有多名员工处理分公司的日常业务，每名员工只能在一个分公司工作。每个分公司由一名经理负责管理分公司的业务和员工，系统需要记录每名经理的任职时间。

(2) 员工信息包括员工工号、姓名、岗位、薪资、手机号和家庭地址。其中，员工号唯一标识员工信息的每一个元组。岗位包括经理、调度员、业务员等。业务员根据客户提交的快件申请单进行快件受理事宜，一个业务员可以受理多个客户的快件申请，一个快件申请只能由一个业务员受理。调度员根据已受理的申请单安排快件的承运事宜，例如：执行承运的业务员、运达时间等。一个业务员可以执行调度员安排的多个快件的承运业务。

(3) 客户信息包括客户号、单位名称、通信地址、所属省份、联系人、联系电话、银行账号。其中，客户号唯一标识客户信息的每一个元组。当客户要寄快件时，先要提交快件申请单，申请号由系统自动生成。快件申请信息包括申请号、客户号、发件人、发件人电

话、快件名称、运费、发出地、收件人、收件人电话、收件地址。其中，一个申请号对应唯一的一个快件申请，一个客户可以提交多个快件申请，但一个快件申请由唯一的一个客户提交。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(见图 2-12)和关系模式(不完整)如下。

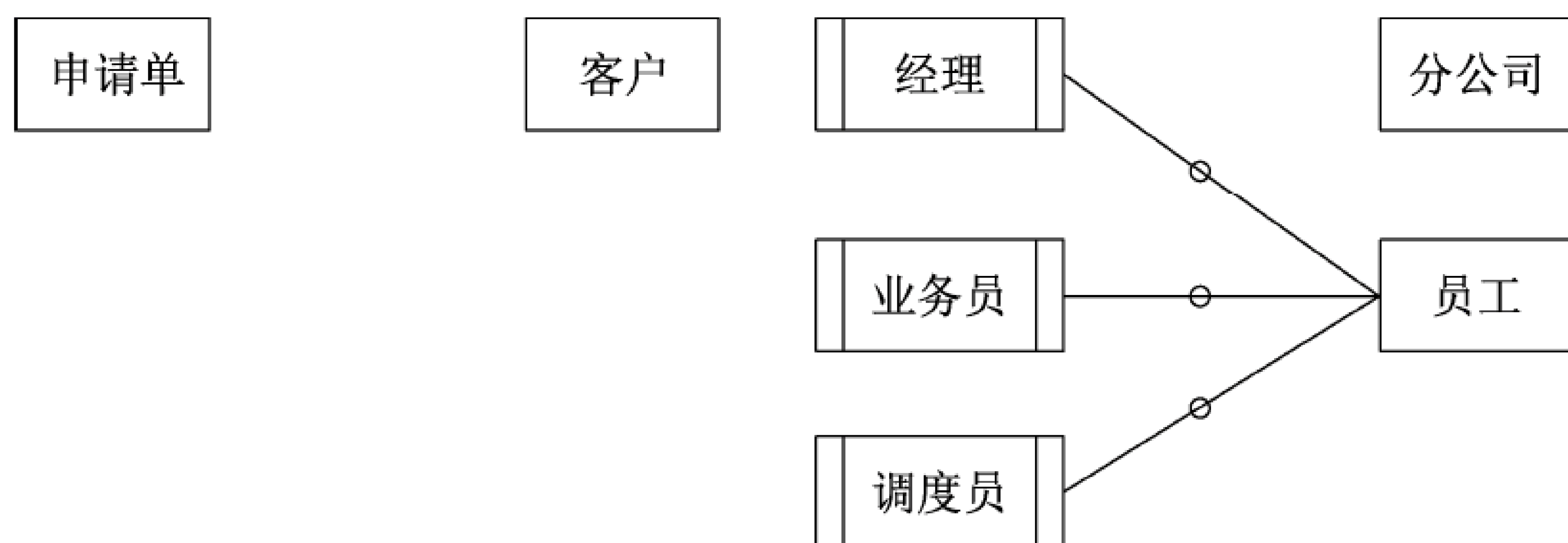


图 2-12　某物品运送信息管理系统实体联系图(不完整)

【关系模式设计】

分公司(分公司编号，名称，经理，办公电话，地址)；

员工(员工号，姓名，(a)，岗位，薪资，手机号，家庭地址)；

客户(客户号，单位名称，通信地址，所属省份，联系人，联系电话，银行账号)；

申请单((b)，发件人，发件人电话，发件人地址，快件名称，运费，收件人，收件人电话，收件地址，受理标志，业务员)；

安排承运((c)，实际完成时间，调度员)。

【问题 1】(5 分)

根据问题描述，补充五个联系，完善图 2-12 的实体联系图。联系名可用联系 1、联系 2、联系 3、联系 4 和联系 5 代替，联系的类型分为 1∶1、1∶n 和 m∶n(或 1∶1、1∶*和*∶*)。

【问题 2】(6 分)

(1) 根据实体联系图，将关系模式中的空(a)~(c)补充完整。

(2) 给出员工、申请单和安排承运关系模式的主键和外键。

【问题 3】(4 分)

(1) 客户关系的通信地址可以进一步分为邮编、省、市、街道，那么该属性是否属于简单属性，为什么？请用 100 字以内的文字说明。

(2) 假设分公司需要增设一位经理的职位，那么分公司与经理之间的联系类型应修改为(d)，分公司的主键应修改为(e)。

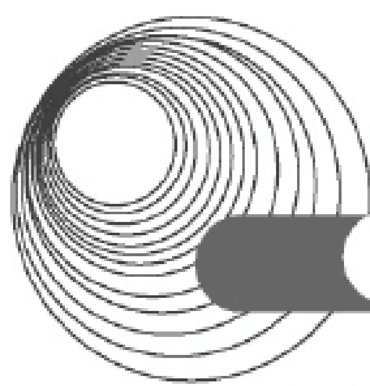
解析：

本题考查数据库设计，涉及的考点有数据库的概念结构设计和逻辑结构设计。

【问题 1】

由“每个分公司有一位经理”可知分公司与经理之间的管理联系类型为 1∶1；由“每个分公司有多名员工处理日常事务，每个员工属于一个分公司”可知分公司与员工间的所属联系类型为 1∶*；并且员工是经理的超类型，经理是员工的子类型。

由“一个客户可以有多个快件申请，但一个快件申请对应唯一的一个客户”可知，客



户与申请单之间的提交联系类型为 1∶*。

由“业务员根据客户提交的快件申请单进行快件受理事宜，一个业务员可以受理多个客户的快件申请，一个快件申请只能由一个业务员受理”可知，业务员与申请单之间的受理联系类型为 1∶*。

由“调度根据已受理的申请单安排快件的承运事宜，例如：执行承运的业务员、运达时间等；一个业务员可以执行调度安排的多个快件的承运业务。”可知，调度、业务员和申请单之间的承运联系类型为 1∶*∶*。

【问题 2】

逻辑结构设计中，分公司信息包括分公司编号、名称、经理、办公电话和地址。每个分公司可以有多名员工处理分公司的日常业务，故主键为员工号；外键为分公司编号。

业务员根据客户提交的快件申请单进行快件受理事宜，一个业务员可以受理多个客户的快件申请，一个快件申请只能由一个业务员受理，故主键为申请号，外键为(客户号、业务员)；调度员根据已受理的申请单安排快件的承运事宜，例如：执行承运的业务员、运达时间等。一个业务员可以执行调度员安排的多个快件的承运业务。故主键为申请号，外键为(业务员、调度员)。

【问题 3】

(1) 简单属性是原子的、不可再分的，复合属性是可以细分为更小的部分。根据题意，客户关系的通信地址可以进一步分为邮编、省、市、街道，该属性属于复合属性。

(2) 分公司需要增设一位经理的职位，分公司可以有多位经理，所以分公司与经理之间的联系类型应该修改为 1∶n，分公司主键应修改为(分公司编号，经理)。

答案：

【问题 1】

完善的实体联系图如图 2-13 所示。

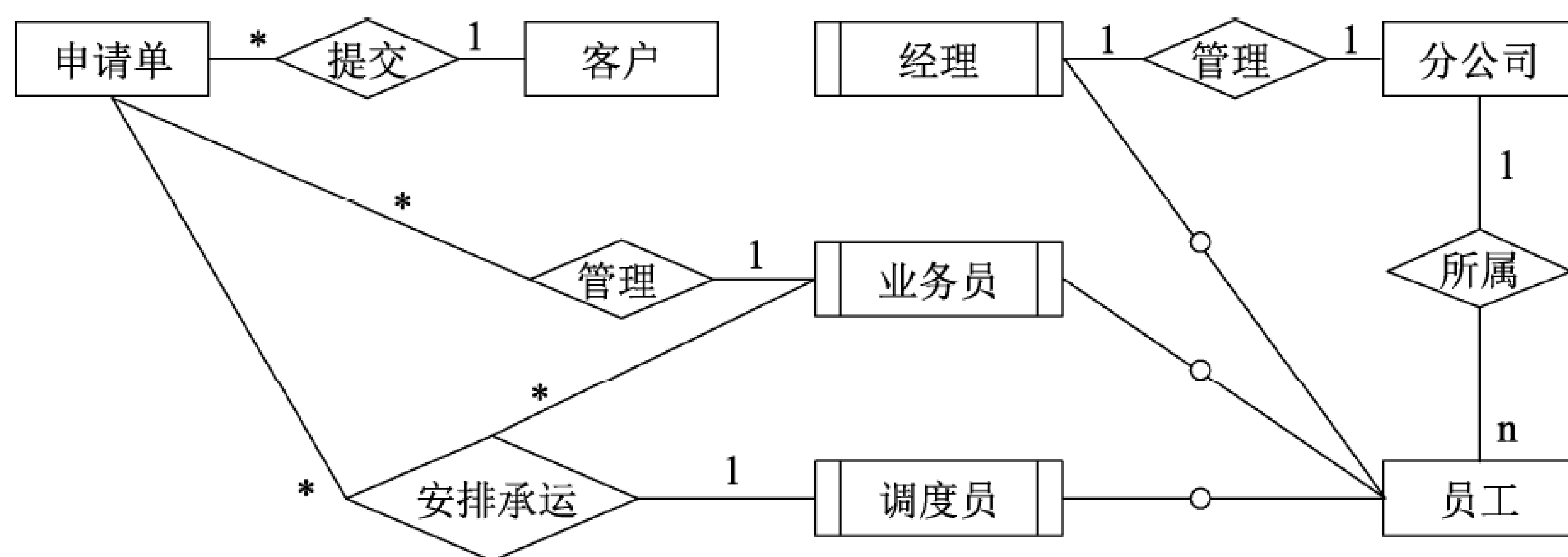


图 2-13　例 6 完善的实体联系图

【问题 2】

(1)

(a) 分公司编号；

(b) 申请号，客户号；

(c) 申请号，业务号。

(2) 关系模式的主键和外键如表 2-1 所示。

表 2-1　关系模式的主键和外键

关系模式	主　键	外　键
员工	员工号	分公司编号
申请单	申请号	客户号，业务员
安排承运	申请号	业务员，调度员

【问题 3】

(1) 该属性不属于简单属性。客户关系的通信地址可以进一步分为邮编、省、市、街道，该属性属于复合属性。

(2) (d)1∶n; (e) 分公司编号，经理。

例 7　阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2013 年 5 月试题二)

【说明】

某电视台拟开发一套信息管理系统，以方便对全台的员工、栏目、广告和演播厅等进行管理。

【需求分析】

(1) 系统需要维护全台员工的详细信息、栏目信息、广告信息和演播厅信息等。员工的信息主要包括工号、姓名、性别、出生日期、电话、住址等。栏目信息主要包括栏目名称、播出时间、时长等。广告信息主要包括广告编号、价格等。演播厅信息主要包括房间号、房间面积等。

(2) 电视台分局调度单用来协调各档栏目、演播厅和场务。一档栏目只会占用一个演播厅，但会使用多名场务来进行演出协调。演播厅和场务可以被多个栏目循环使用。

(3) 电视台根据栏目来插播广告。每档栏目可以插播多条广告，每条广告可以在多档栏目中插播。

(4) 一档栏目可以有多名主持人，但一名主持人只能主持一档栏目。

(5) 一名编辑人员可以编辑多条广告，一条广告只能由一名编辑人员编辑。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-14 所示。

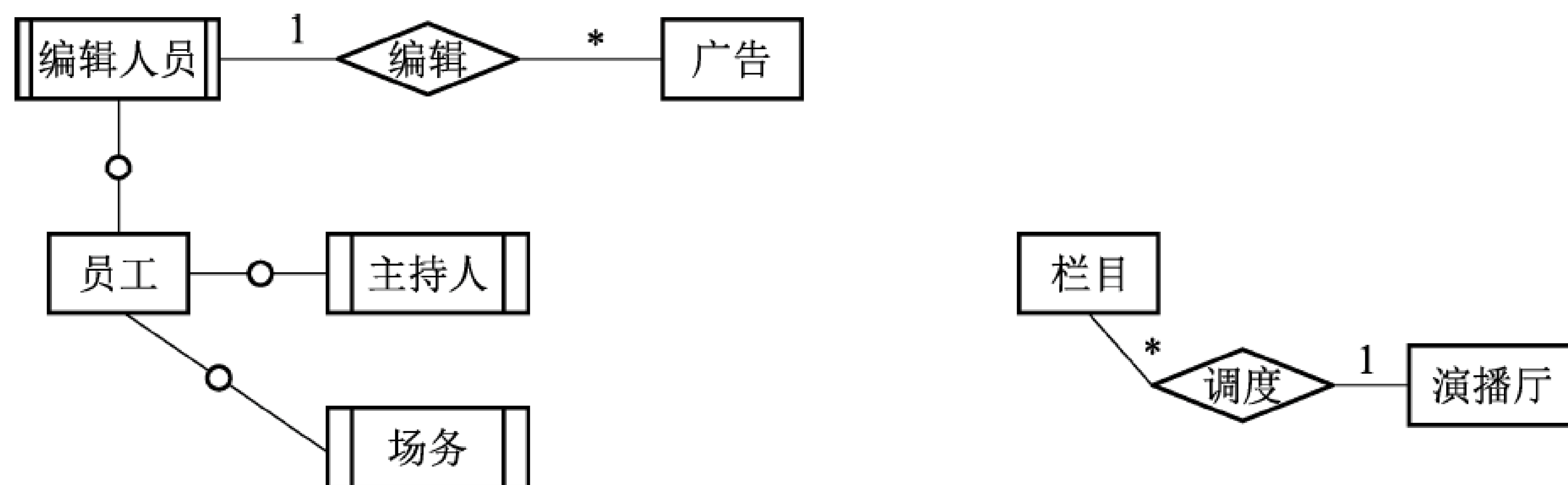


图 2-14　例 7 实体联系图(不完整)

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)。

演播厅(<u>房间号</u>，房间面积)；

栏目(<u>栏目名称</u>，播出时间，时长)；

广告(广告编号，销售价格，______(1)______)；

员工(<u>工号</u>，姓名，性别，出生日期，电话，住址)；

主持人(主持人工号，______(2)______)；

插播单(______(3)______，播出时间)；

调度单(______(4)______)。

【问题 1】(7 分)

补充图 2-14 中的联系和联系的类型。

【问题 2】(5 分)

根据图 2-14，将逻辑结构设计阶段生成的关系模式中的空(1)~(4)补充完整，并用下画线指出(1)~(4)所在关系模式的主键。

【问题 3】(3 分)

现需要记录广告商信息，增加广告商实体。一个广告商可以提供多条广告，一条广告只由一个广告商提供。请根据该要求，对图 2-14 进行修改，画出修改后的实体间联系和联系的类型。

解析：

本题考查数据库设计，涉及的考点有数据库的概念结构设计和逻辑结构设计。两个实体集之间的联系类型分为三类：一对一(1∶1)联系、一对多(1∶n)联系和多对多(m∶n)联系。

【问题 1】

根据题意，每档栏目可以插播多条广告，每条广告可以在多档栏目中插播，因此广告和栏目之间是名为“插播”的多对多联系。一名主持人只能主持一档栏目，一档栏目可以被多名主持人所主持，因此栏目和主持人之间是名为“主持”的一对多联系；一档栏目会使用多名场务来进行演出协调，而一名场务可以被多个栏目循环使用，因此场务与栏目之间是名为“使用”的多对多联系。

【问题 2】

逻辑结构设计中，广告实体中缺少广告时长，主键为广告编号；栏目实体与主持人实体间存在一对多联系，故将栏目的主键栏目名称加入到主持人实体中，主键为主持人工号；插播单为栏目实体和广告实体间的多对多联系所派生的实体，其中记录了栏目和广告的主键信息，故插播单中缺少栏目名称和广告编号信息，主键为栏目名称和广告编号；调度单为场务、栏目和演播厅实体间的多对多联系所派生的实体，故其记录了栏目名称、房间号、场务工号，主键为栏目名称和场务工号。

【问题 3】

因为一个广告商可以提供多条广告，一条广告只能由一个广告商提供，故广告商和广告之间存在一对多联系。

答案：

【问题 1】

补充后的实体联系图如图 2-15 所示。

【问题 2】

(1) 广告时长。主键：广告编号。

(2) 栏目名称。主键：主持人工号。

(3) 栏目名称，广告编号。主键：栏目名称，广告编号。

(4) 栏目名称，房间号，场务工号。主键：栏目名称，场务工号。

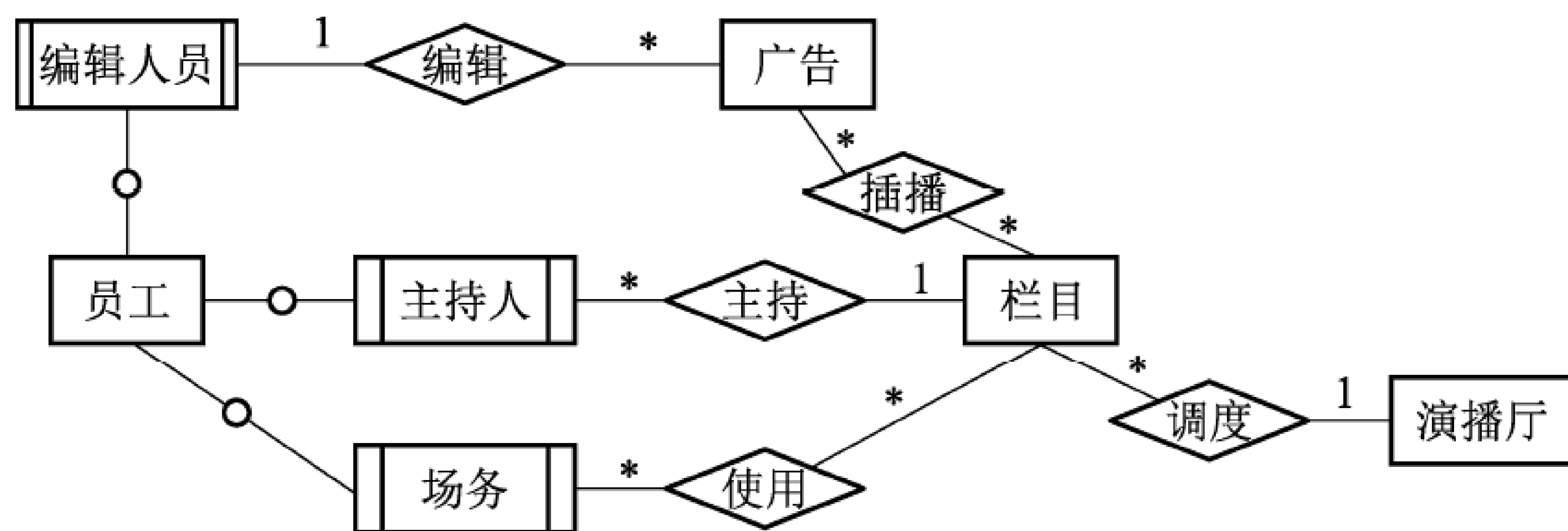


图 2-15　例 7 补充后的实体联系图

【问题 3】

修改后的实体联系图如图 2-16 所示。

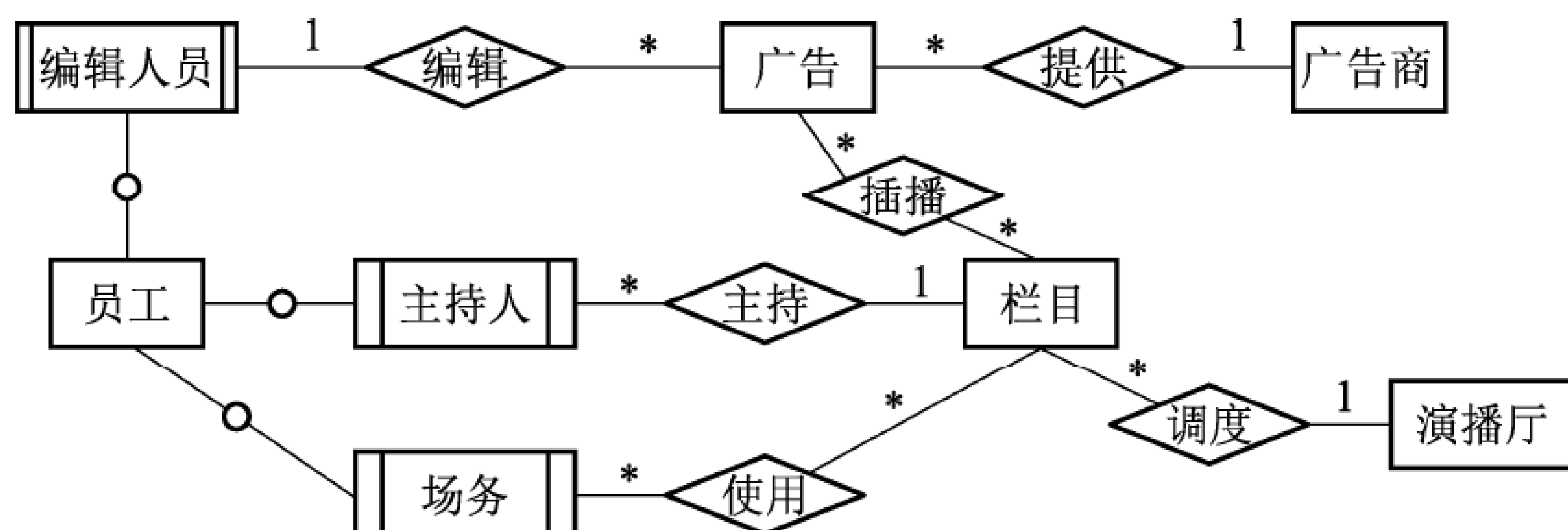


图 2-16　例 7 修改后的实体联系图

2.1.3　同步练习

1. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2012 年 11 月试题二)

【说明】

某会议策划公司为了方便客户，便于开展和管理各项业务活动，需要构建一个基于网络的会议预定系统。

【需求分析】

(1) 会议策划公司设有受理部、策划部和其他部门，部门信息包括部门号、部门名称、部门主管、电话和邮箱号，每个部门有多名员工处理部门的日常事务，每名员工只能在一个部门工作，每个部门有一名主管负责管理本部门的事务和人员。

(2) 员工信息包括员工号、姓名、部门号、职位、联系方式和工资；其中，职位包括主管、业务员、策划员等。业务员负责受理会议申请，若申请符合公司规定，则设置受理标

志并填写业务员的员工号。策划部主管为已受理的会议申请制定策划任务，包括策划内容、参与人数、要求完成时间等。一个已受理的会议申请对应一个策划任务，一个策划任务只对应一个已受理的会议申请，但一个策划任务可由多名策划员参与执行，且一名策划员可以参与多项策划任务。

(3) 客户信息包括客户号、单位名称、通信地址、所属省份、联系人、联系电话、银行账号。其中，一个客户号唯一标识一个客户。一个客户可以提交多个会议申请，但一个会议申请对应唯一的一个客户号。

(4) 会议申请信息包括申请号、开会日期、会议地点、持续天数、会议人数、预算费用、会议类型、酒店要求、会议室要求、客房类型、客房数、联系人、联系方式、受理标志和业务员的员工号等。客房类型有豪华套房、普通套房、标准间、三人间等，且申请号和客房类型决定客房数。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图2-17所示。

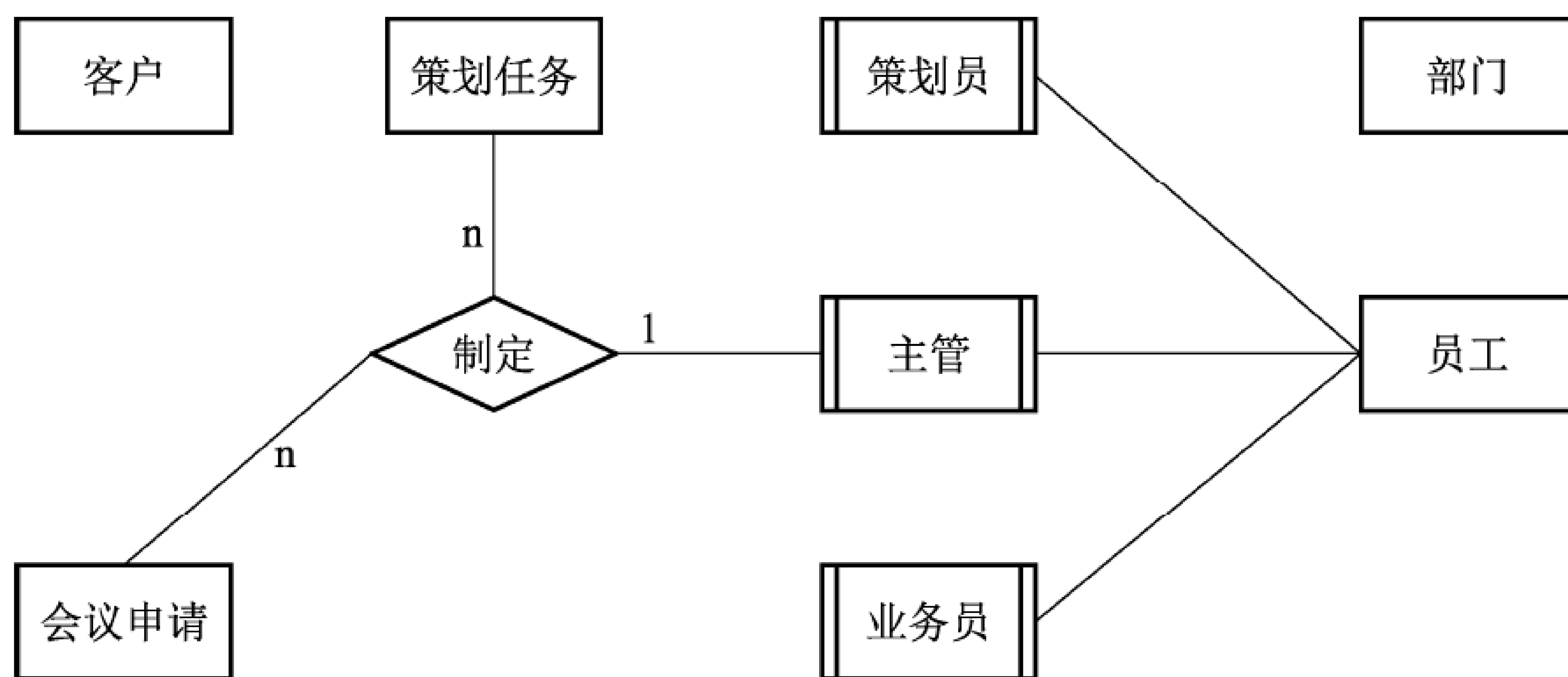


图2-17　实体联系图

【关系模式设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)。

部门(部门号，部门名称，主管，电话，邮箱号)；

员工(员工号，姓名，__(a)__，联系方式，工资)；

客户(客户号，单位名称，通信地址，所属省份，联系人，联系电话，银行账号)；

会议申请(__(b)__，开会日期，会议地点，持续天数，会议人数，预算费用，会议类型，酒店要求，会议室要求，客房数，联系人，联系方式，受理标志，员工号)；

策划任务(__(c)__，策划内容，参与人数，要求完成时间)；

执行策划(__(d)__，实际完成时间)。

【问题1】(5分)

根据问题描述，补充五个联系及联系的类型，完善图2-17所示的实体联系图。

【问题2】(7分)

根据实体联系图，将关系模式中的空(a)~(d)补充完整(1个空缺处可能有多个数据项)。

对会议申请、策划任务和执行策划关系模式，用下画线和#分别指出各关系模式的主键和外键。

【问题 3】(3 分)

请说明会议申请关系模式存在的问题及解决方案。

2. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2012 年 5 月试题二)

【说明】

某医院拟开发一套住院病人信息管理系统，以方便对住院病人、医生、护士和手术等信息进行管理。

【需求分析】

(1) 系统登记每个病人的住院信息，包括病案号、病人的姓名、性别、地址、身份证号、电话号码、入院时间及病床号信息，每个病床有唯一所属的病区及病房，如表 2-2 所示。其中病案号唯一标识病人本次住院的信息。

表 2-2　住院登记表

病案号	071002286	姓名	张三	性别	男
身份证号	0102196701011234	入院时间	2011-03-03	病床号	052401
病房	0524 室	病房类型	三人间	所属病区	05 II 区

(2) 在一个病人的一次住院期间，由一名医生对该病人的病情进行诊断，并填写一份诊断书，如表 2-3 所示。对于需要进行一次或多次手术的病人，系统记录手术名称、手术室、手术日期、手术时间、主刀医生及多名协助医生。每名医生在手术中的责任不同，如表 2-4 所示，其中手术室包含手术室号、楼层、地点和类型等信息。

表 2-3　诊断书

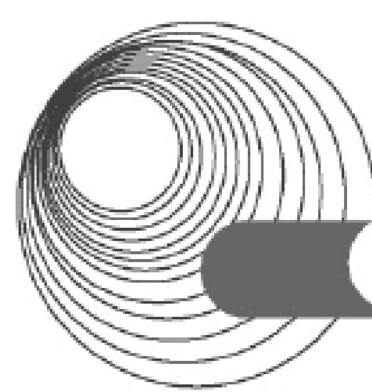
诊断时间：2011 年 3 月

病案号	071002286	姓名	张三	性别	男	医生	李**
诊断							

表 2-4　手术安排表

手术名称	**手术	病案号	071002286	姓名	张三	性别	男
手术室	032501	手术日期	2011-03-15	手术时间	8:30~10:30	主刀医生	李**
协助医生	王**(协助)，周**(协助)，刘**(协助)，高**(麻醉)						

(3) 护士分为两类：病床护士和手术室护士。每个病床护士负责护理一个病区内的所有病人，每个病区由多名护士负责护理。手术室护士负责手术室的护理工作。每个手术室护士负责多个手术室，每个手术室由多名护士负责，每个手术室护士在手术室中有不同的责



任，并由系统记录其责任。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-18 所示。

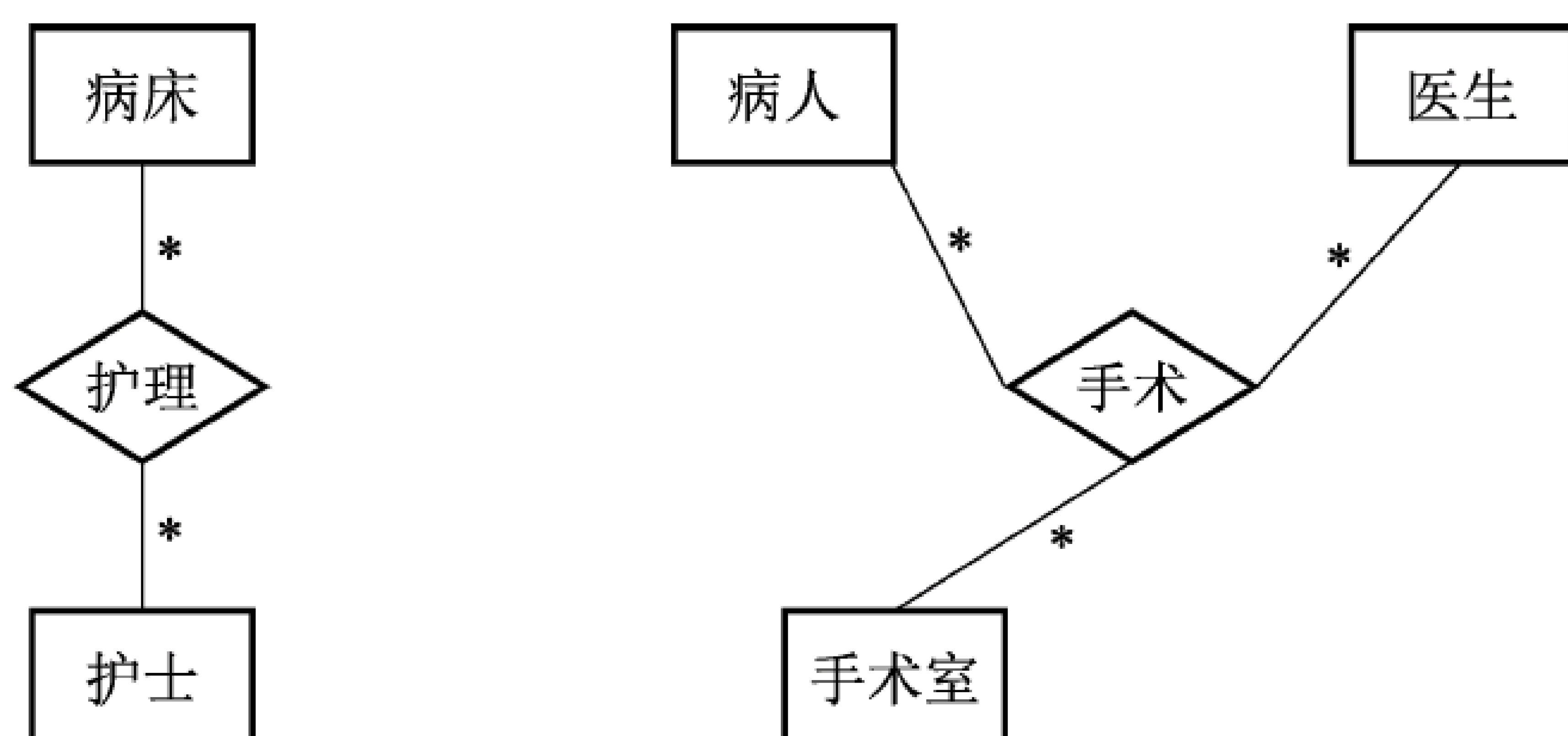


图 2-18　实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)。

病床(病床号，病房，病房类型，所属病区)；

护士(护士编号，姓名，类型，性别，级别)；

病床护士((1))；

手术室(手术室号，楼层，地点，类型)；

手术室护士((2))；

病人((3) ，姓名，性别，地址，身份证号，电话号码，入院时间)；

医生(医生编号，姓名，性别，职称，所属科室)；

诊断书((4) ，诊断，诊断时间)；

手术安排(病案号，手术室号，手术时间，手术名称)；

手术医生安排((5) ，医生责任)。

【问题 1】(6 分)

补充图 2-18 中的联系和联系的类型。

【问题 2】(5 分)

根据图 2-18，将逻辑结构设计阶段生成的关系模式中的空(1)~(5)补充完整，并用下画线指出主键。

【问题 3】(4 分)

如果系统还需要记录医生给病人的用药情况，即记录医生给病人所开处方中药品的名称、用量、价格、药品的生产厂家等信息，请根据该要求，对图 2-18 进行修改，画出补充后的实体、实体间联系和联系的类型。

3. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2011 年 11 月试题二)

【说明】

某物流公司为了整合上游供应商与下游客户，缩短物流过程，降低产品库存，需要构建一个信息系统，以方便管理其业务运作活动。

【需求分析结果】

(1) 物流公司包含若干部门，部门信息包括部门号、部门名称、经理、电话和邮箱。一个部门可以有多名员工处理部门的日常事务，每名员工只能在一个部门工作。每个部门有一名经理，只需负责本部门的事务和人员。

(2) 员工信息包括员工号、姓名、职位、电话号码和工资。其中，职位包括经理、业务员等。业务员根据托运申请负责安排承运货物事宜，如装货时间、到达时间等。一个业务员可以安排多个托运申请，但一个托运申请只由一个业务员处理。

(3) 客户信息包括客户号、单位名称、通信地址、所属省份、联系人、联系电话、银行账号。其中，客户号唯一标识客户信息的每一个元组。每当客户要进行货物托运时，先要提出货物托运申请。托运申请包括申请号、客户号、货物名称、数量、运费、出发地、目的地。其中，一个申请号对应唯一的一个托运申请；一个客户可以有多个货物托运申请，但一个托运申请对应唯一的一个客户号。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-19 所示。

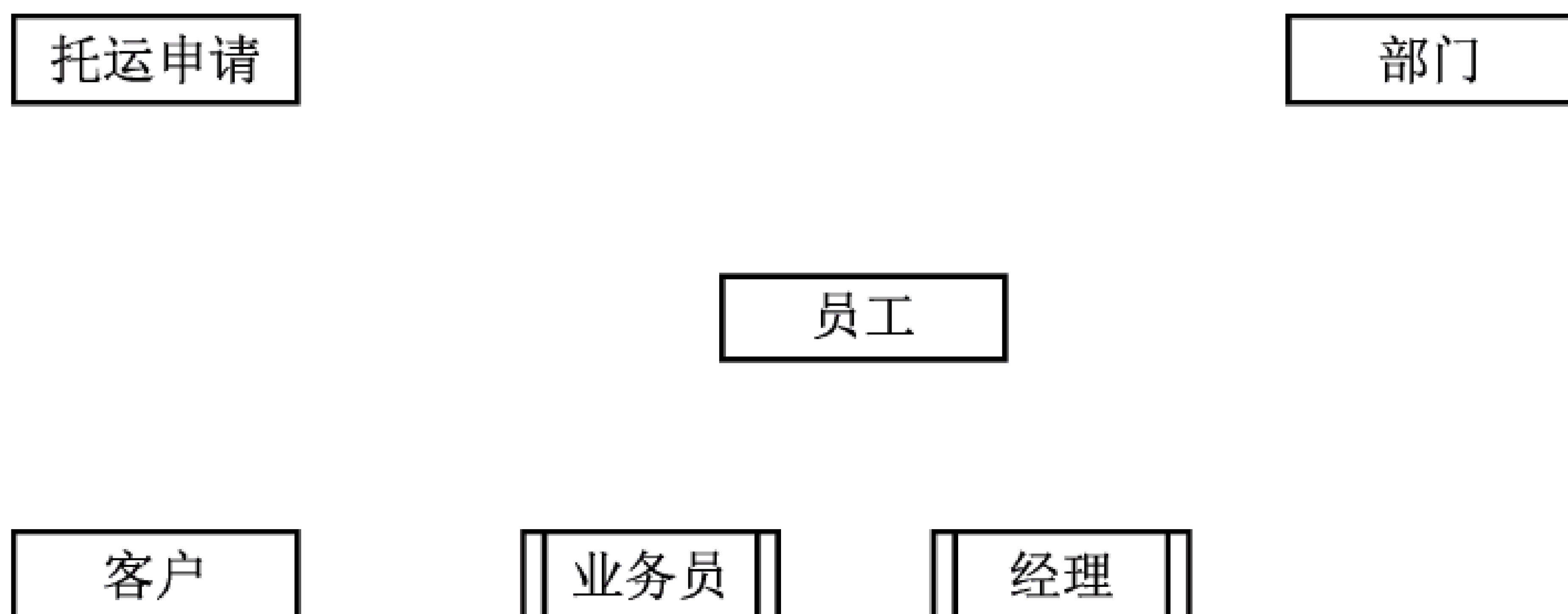


图 2-19　实体联系图

【关系模式设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

部门(部门号，部门名称，经理，电话，邮箱)；

员工(员工号，姓名，职位，电话号码，工资，(a))；

客户((b)，单位名称，通信地址，所属省份，联系人，联系电话，银行账号)；

托运请求((c)，货物名称，数量，运费，出发地，目的地)；

安排承运((d)，装货时间，到达时间，业务员)。

【问题 1】(5 分)

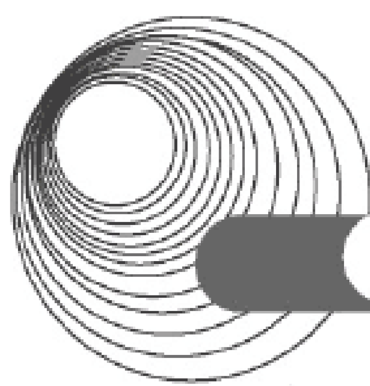
根据问题描述，补充四个联系和联系的类型，以及实体与子实体的联系，完善图 2-19 所示的实体联系图。

【问题 2】(8 分)

根据实体联系图，将关系模式中的空(a)~(d)补充完整。分别指出部门、员工和安排承运关系模式的主键和外键。

【问题 3】(2 分)

若系统新增需求描述如下：为了数据库信息的安全性，公司要求对数据库操作设置权限管理功能，当员工登录系统时，系统需要检查员工的权限。权限的设置人是部门经理。



为了满足上述需求，应如何修改(或补充)图 2-19 所示的实体联系图，请给出修改后的实体联系图和关系模式。

4. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2011 年 5 月试题二)

【说明】

某服装销售公司拟开发一套服装采购管理系统，以便对服装采购和库存进行管理。

【需求分析结果】

(1) 采购系统需要维护服装信息及服装在仓库中的存放情况。服装信息主要包括服装编码、服装描述、服装类型、销售价格、尺码和面料。其中，服装类型为销售分类，服装按销售分类编码。仓库信息包括仓库编码、仓库位置、仓库容量和库管员。系统记录库管员的库管员编码、姓名和级别。一个库管员可以管理多个仓库，每个仓库有一名库管员。一个仓库中可以存放多类服装，一类服装可能存放在多个仓库中。

(2) 当库管员发现一类或者多类服装缺货时，需要生成采购订单。一个采购订单可以包含多类服装。每类服装可由多个不同的供应商供应，但具有相同的服装编码。采购订单主要记录订单编码、订货日期和应到货日期，并详细记录所采购的每类服装的数量、采购价格和对应的多个供应商。

(3) 系统需记录每类服装的各个供应商信息和供应情况。供应商信息包括供应商编码、供应商名称、地址、企业法人和联系电话。供应情况记录供应商所供应服装的服装类型和服装质量等级。一个供应商可以供应多类服装，一类服装可由多个供应商供应。库管员根据入库时的服装质量情况，设定或修改每个供应商所供应的每类服装的服装质量等级，作为后续采购服装时选择供应商的参考标准。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-20 所示。

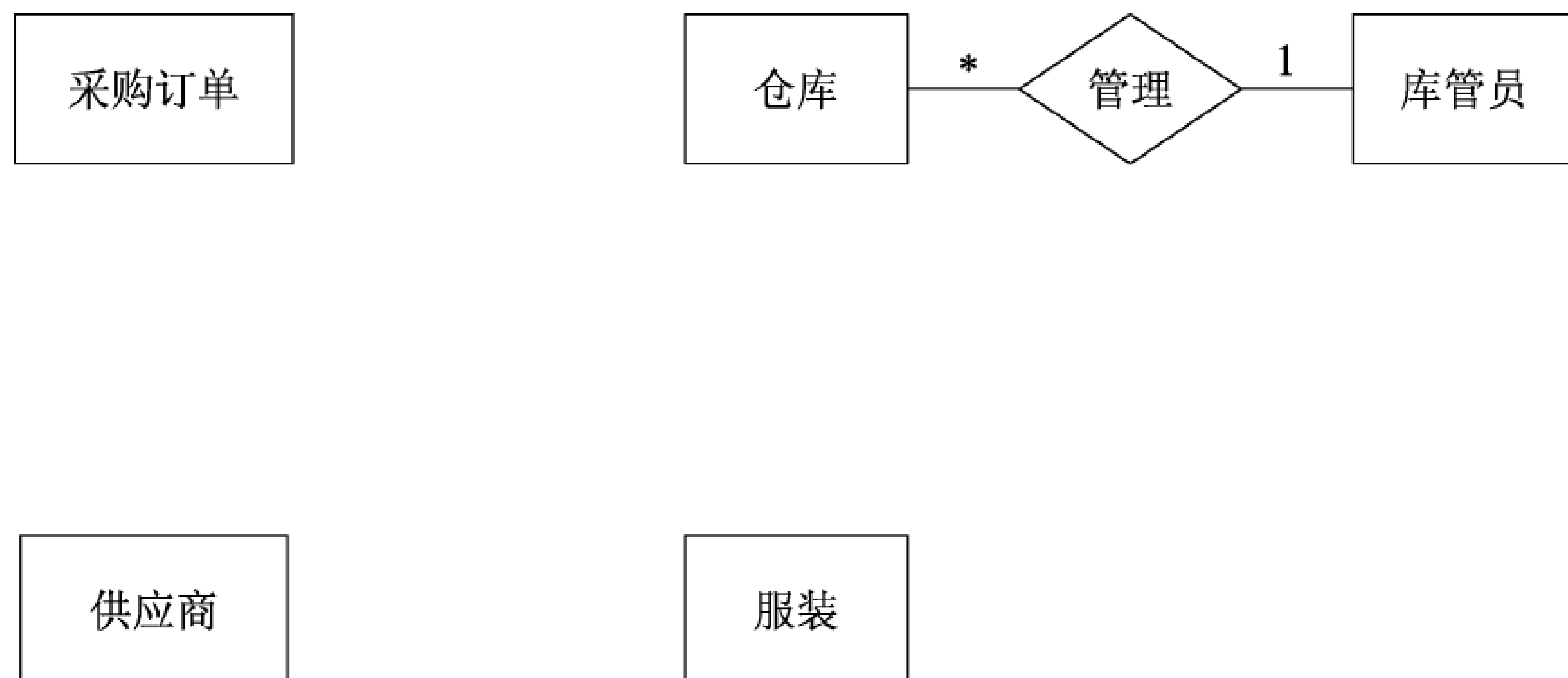


图 2-20　实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

库管员(库管员编码，姓名，级别)；

仓库信息((1) ，仓库位置，仓库容量)；

服装(服装编码，服装描述，服装类型，尺码，面料，销售价格)；
供应商(供应商编码，供应商名称，地址，联系电话，企业法人)；
供应情况((2) ，服装质量等级)；
采购订单((3))；
采购订单明细((4))。

【问题 1】(6 分)

根据需求分析的描述，补充图 2-20 中的联系和联系的类型。

【问题 2】(6 分)

根据补充完整的图 2-20，将逻辑结构设计阶段生成的关系模式中的空(1)~(4)补充完整，并给出其主键(用下画线指出)。

【问题 3】(3 分)

如果库管员定期需要轮流对所有仓库中的服装质量进行抽查，对每个仓库中的每一类被抽查服装需要记录一条抽查结果，并且需要记录抽查的时间和负责抽查的库管员。请根据该要求对图 2-20 进行修改，画出修改后的实体联系图和联系的类型。

5. 阅读以下说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2010 年 11 月试题二)

【说明】

某公司拟开发一套小区物业收费管理系统。初步的需求分析结果如下。

(1) 业主信息主要包括业主编号、姓名、房号、房屋面积、工作单位、联系电话等。房号可唯一标识一条业主信息，且一个房号仅对应一套房屋；一个业主可以有一套或多套的房屋。

(2) 部门信息主要包括部门号、部门名称、部门负责人、部门电话等。一个员工只能属于一个部门，一个部门只有一位负责人。

(3) 员工信息主要包括员工号、姓名、出生年月、性别、住址、联系电话、所在部门号、职务和密码等。根据职务不同，员工可以有不同的权限，职务为“经理”的员工具有更改(添加、删除和修改)员工表中本部门员工信息的操作权限；职务为“收费”的员工只具有收费的操作权限。

(4) 收费信息包括房号、业主编号、收费日期、收费类型、数量、收费金额、员工号等。收费类型包括物业费、卫生费、水费和电费，并按月收取，收费标准如表 2-5 所示。其中：物业费=房屋面积(平方米)×每平方米单价，卫生费=套房数量(套)×每套房单价，水费=用水数量(吨)×每吨水单价，电费=用电数量(度)×每度电单价。

表 2-5　收费标准

收费类型	单　位	单价/元
物业费	平方米	1.00
卫生费	套	10.00
水　费	吨	0.70
电　费	度	0.80

(5) 收费完毕后系统应为业主生成收费单，收费单示例如表2-6所示。

表 2-6　收费单示例

房号：A1608　　　　　　　　　　　　　　　　　　　　　　　　　　　业主姓名：李斌

序　号	收费类型	数　量	金额/元
1	物业费	98.6	98.6
2	卫生费	1	10.00
3	水费	6	4.20
4	电费	102	81.60
合计	壹佰玖拾肆元零肆角		194.40

收费日期：2010-9-2　　　　　　　　　　　　　　　　　　　　　员工号：001

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图2-21所示。在图2-21中，收费员和经理是员工的子实体。

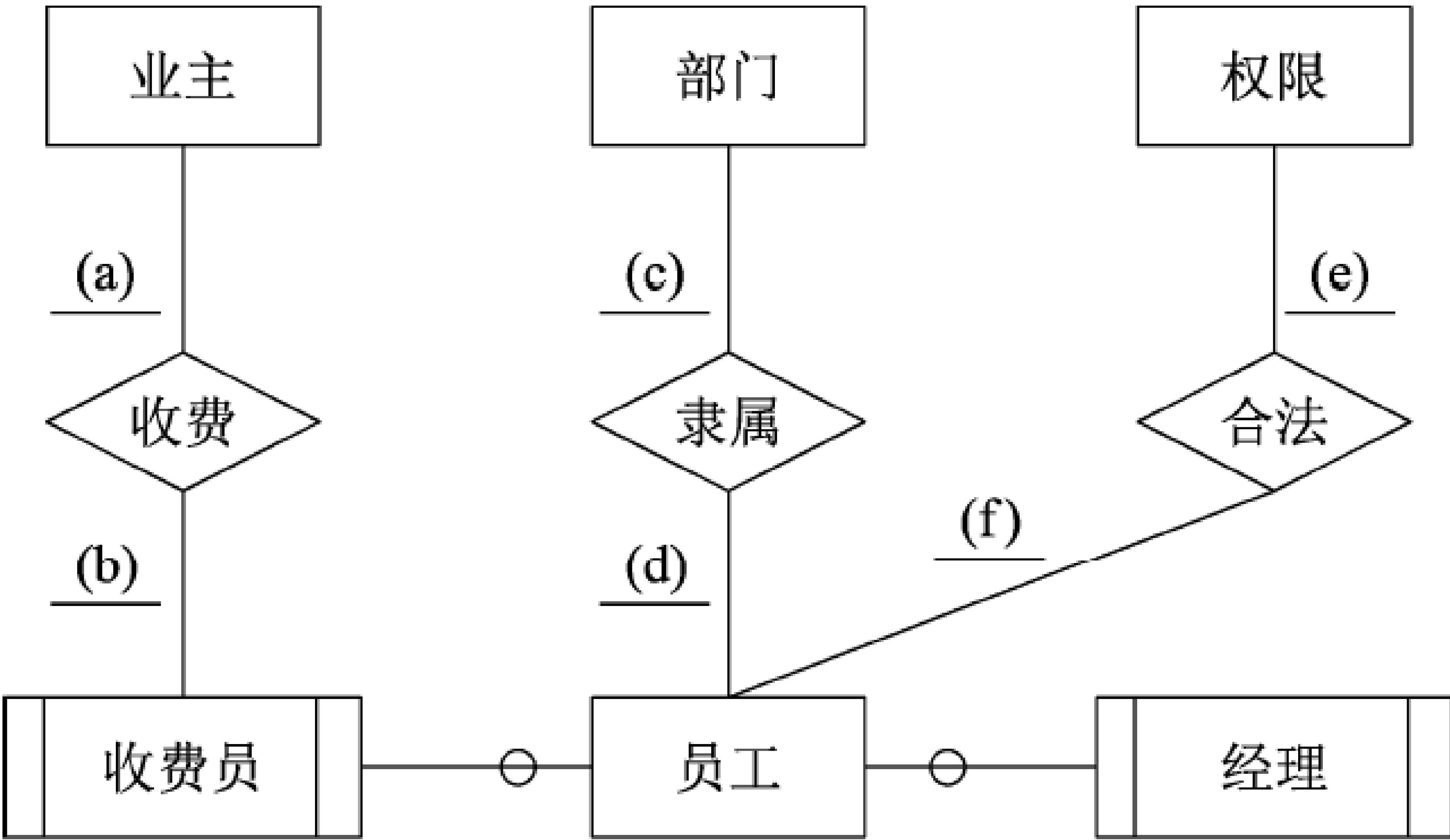


图 2-21　实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

业主((1) ，姓名，房屋面积，工作单位，联系电话)；

员工((2) ，姓名，出生年月，性别，住址，联系电话，职务，密码)；

部门((3) ，部门名称，部门电话)；

权限(职务，操作权限)；

收费标准((4))；

收费信息((5) ，收费类型，收费金额，员工号)。

【问题1】(8分)

根据图2-21，将逻辑结构设计阶段生成的关系模式中的空(1)~(5)补充完整，然后给出各关系模式的主键和外键。

【问题2】(5分)

填写图2-21中(a)~(f)处联系的类型(注：一方用1表示，多方用m或n或*表示)，并将

图 2-21 中的实体、联系和联系的类型补充完整。

【问题 3】(2 分)

业主关系模式属于第几范式？请说明存在的问题。

6. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2010 年 5 月试题二)

【说明】

某学校拟开发一套实验管理系统，对各课程的实验安排情况进行管理。

【需求分析结果】

一个实验室可进行多种类型不同的实验。由于实验室和实验员资源有限，需根据学生人数分批次安排实验室和实验员。一门课程可以为多个班级开设，每个班级每学期可以开设多门课程。一门课程的一种实验可以根据人数、实验室的可容纳人数和实验类型，分批次开设在多个实验室的不同时间段。一个实验室的一次实验可以分配多个实验员负责辅导实验，实验员给出学生的每次实验成绩。

(1) 课程信息包括课程编号、课程名称、实验学时、授课院系和开课的班级等信息；实验信息记录该课程的实验进度信息，包括实验名、实验类型、学时、安排周次等信息，如表 2-7 所示。

表 2-7 课程及实验信息

<table>
<tr><th>课程编号</th><td>15054037</td><th>课程名称</th><td colspan="2">数字电视原理</td><th>实验学时</th><td>12</td></tr>
<tr><th>班 级</th><td>电 0501，信 0501，计 0501</td><th>授课院系</th><td colspan="2">机械与电气工程</td><th>授课学期</th><td>第三学期</td></tr>
<tr><th>序 号</th><th colspan="2">实 验 名</th><th>实验类型</th><th>难 度</th><th>学 时</th><th>安排周次</th></tr>
<tr><td>1505403701</td><td colspan="2">音视频 AD-Da 实验</td><td>验证性</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>1505403702</td><td colspan="2">音频编码实验</td><td>验证性</td><td>2</td><td>2</td><td>5</td></tr>
<tr><td>1505403703</td><td colspan="2">视频编码实验</td><td>演示性</td><td>0.5</td><td>1</td><td>9</td></tr>
</table>

(2) 以课程为单位制定实验安排计划信息，包括实验地点、实验时间、实验员等信息，如表 2-8 所示。

表 2-8 实验安排计划

<table>
<tr><th>课程编号</th><td>15054037</td><th>课程名称</th><td>数字电视原理</td><th>安排学期</th><td>2009 年秋</td><th>总人数</th><td>220</td></tr>
<tr><th>实验编号</th><th colspan="2">实 验 名</th><th>实 验 员</th><th>实验时间</th><th>地 点</th><th>批次号</th><th>人数</th></tr>
<tr><td>1505403701</td><td colspan="2">音视频 AD-DA 实验</td><td>盛×，陈×</td><td>第 3 周周四晚上</td><td>实验三楼 310</td><td>1</td><td>60</td></tr>
<tr><td>1505403701</td><td colspan="2">音视频 AD-DA 实验</td><td>盛×，陈×</td><td>第 3 周周四晚上</td><td>实验三楼 310</td><td>2</td><td>60</td></tr>
<tr><td>1505403701</td><td colspan="2">音视频 AD-DA 实验</td><td>吴×，刘×</td><td>第 3 周周五晚上</td><td>实验三楼 311</td><td>3</td><td>60</td></tr>
<tr><td>1505403701</td><td colspan="2">音视频 AD-DA 实验</td><td>吴×</td><td>第 3 周周五晚上</td><td>实验三楼 311</td><td>4</td><td>40</td></tr>
<tr><td>1505403702</td><td colspan="2">音频编码实验</td><td>盛×，刘×</td><td>第 5 周周一下午</td><td>实验四楼 410</td><td>1</td><td>70</td></tr>
</table>

(3) 由实验员给出每个学生每次实验的成绩，包括实验名、学号、姓名、班级、实验成绩等信息，如表 2-9 所示。

表 2-9　实验成绩

实验员：　盛×

实 验 名	音视频 AD-DA 实验	课 程 名	数字电视原理
学　号	姓　名	班　级	实验成绩
030501001	陈民	信 0501	87
030501002	刘志	信 0501	78
040501001	张勤	计 0501	86

(4) 学生的实验课程总成绩根据每次实验的成绩以及每次实验的难度来计算。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-22 所示。

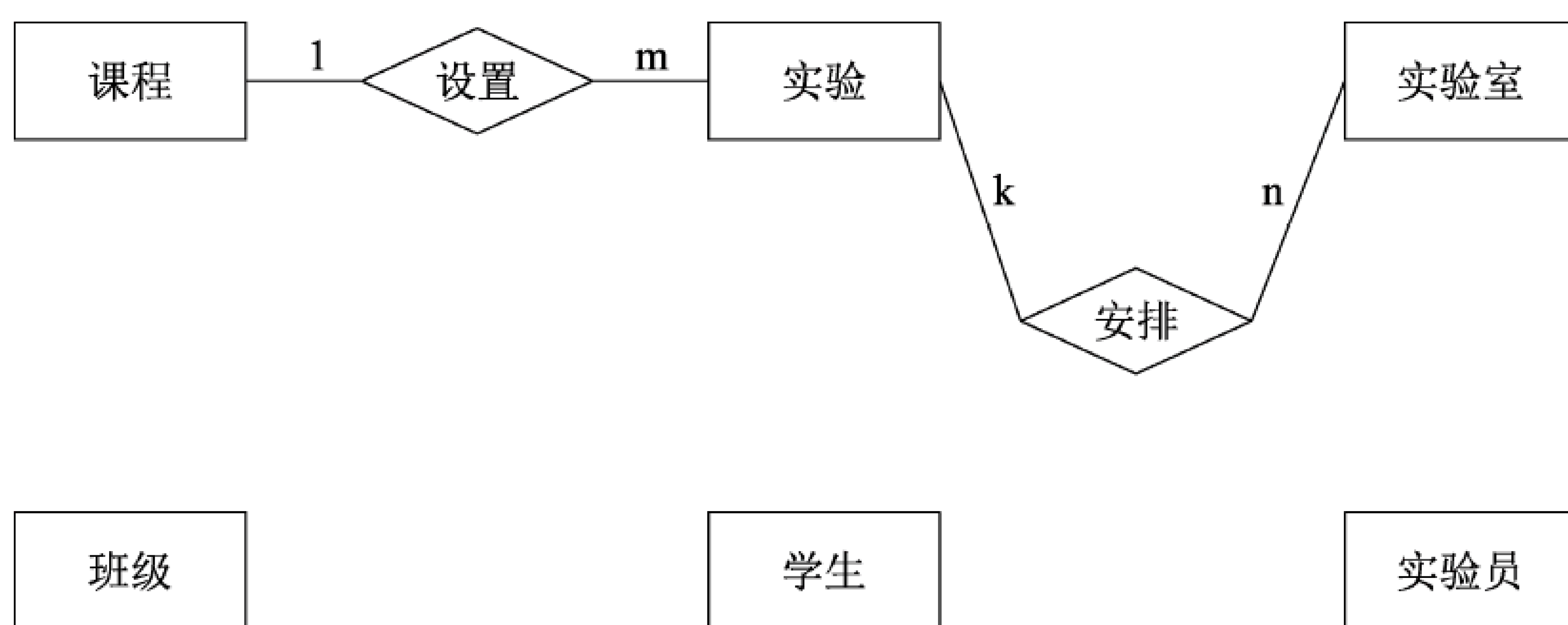


图 2-22　实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

课程(课程编号，课程名称，授课院系，实验学时)；

班级(班级号，专业，所属系)；

开课情况(__(1)__，授课学期)；

实验(__(2)__，实验类型，难度，学时，安排周次)；

实验计划(__(3)__，实验时间，人数)；

实验员(__(4)__，级别)；

实验室(实验室编号，地点，开放时间，可容纳人数，实验类型)；

学生(__(5)__，姓名，年龄，性别)；

实验成绩(__(6)__，实验成绩，评分实验员)。

【问题 1】(6 分)

补充图 2-22 中的联系和联系的类型。

【问题 2】(6 分)

根据图 2-22，将逻辑结构设计阶段生成的关系模式中的空(1)~(6)补充完整，并用下画线指出这六个关系模式的主键。

【问题 3】(3 分)

如果需要记录课程的授课教师，新增加“授课教师”实体，请对图 2-22 进行修改，并

画出修改后的实体间联系和联系的类型。

7. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2009 年 11 月试题二)

【说明】

某公司拟开发一多用户电子邮件客户端系统，部分功能的初步需求分析结果如下。

(1) 邮件客户端系统支持多个用户，用户信息主要包括用户名和用户密码，且系统中的用户名不可重复。

(2) 邮件账号信息包括邮件地址及其相应的密码，一个用户可以拥有多个邮件地址(如userl@123.com)。

(3) 一个用户可拥有一个地址簿，地址簿信息包括联系人编号、姓名、电话、单位地址、邮件地址 1、邮件地址 2、邮件地址 3 等信息。地址簿中一个联系人只能属于一个用户，且联系人编号唯一标识一个联系人。

(4) 一个邮件账号可以含有多封邮件，一封邮件可以含有多个附件。邮件主要包括邮件号、发件人地址、收件人地址、邮件状态、邮件主题、邮件内容、发送时间、接收时间。其中，邮件号在整个系统内唯一标识一封邮件，邮件状态有已接收、待发送、已发送和已删除 4 种，分别表示邮件是属于收件箱、发件箱、已发送箱和废件箱。一封邮件可以发送给多个用户。附件信息主要包括附件号、附件文件名、附件大小。一个附件只属于一封邮件，且附件号仅在一封邮件内唯一。

【问题 1】

根据以上说明设计的 E-R 图如图 2-23 所示，请指出地址簿与用户、邮件账号与邮件、邮件与附件之间的联系类型。

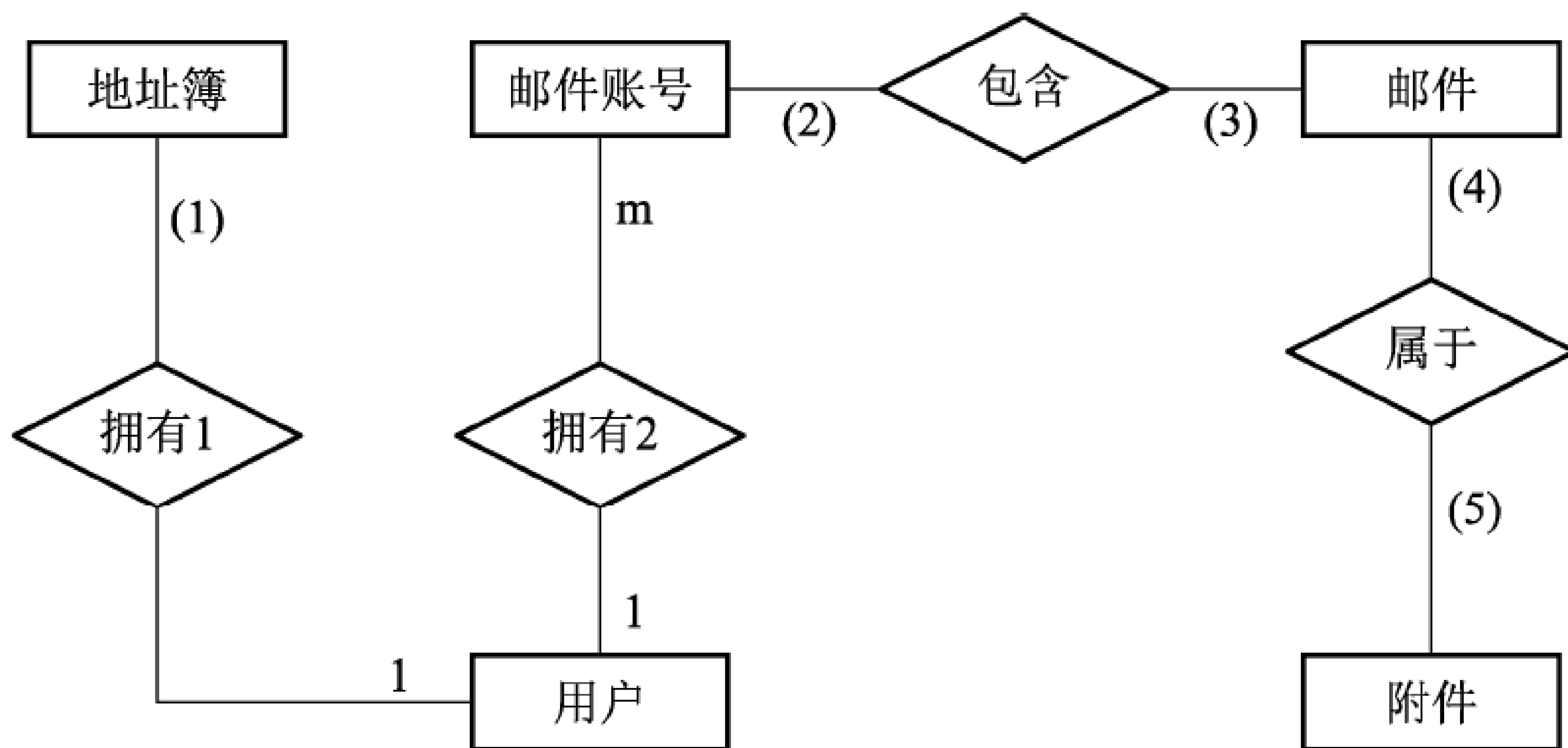


图 2-23　电子邮件客户端系统的 E-R 图

【问题 2】

该邮件客户端系统的主要关系模式如下，请填补(a)~(c)的空缺部分。

用户(用户名，用户密码);

地址簿((a) ，联系人编号，姓名，电话，单位地址，邮件地址 1，邮件地址 2，邮件地址 3);

邮件账号(邮件地址，邮件密码，用户名);

邮件(<u>(b)</u>，收件人地址，邮件状态，邮件主题，邮件内容，发送时间，接收时间)；

附件(<u>(c)</u>，附件号，附件文件名，附件大小)。

【问题 3】

(1) 请指出问题 2 中给出的地址簿、邮件和附件关系模式的主键，如果关系模式存在外键，请指出。

(2) 附件属于弱实体吗？请用 50 字以内的文字说明原因。

8. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2009 年 5 月试题二)

【说明】

某集团公司拥有多个大型连锁商场，公司需要构建一个数据库系统以方便管理其业务运作活动。

【需求分析结果】

(1) 商场需要记录的信息包括商场编号(编号唯一)、商场名称、地址和联系电话。某商场信息如表 2-10 所示。

表 2-10　商场信息表

商场编号	商场名称	地　址	联系电话
PS2101	淮海商场	淮海中路 918 号	021-64158818
PS2902	西大街商场	西大街时代盛典大厦	029-87283229
PS2903	东大街商场	碑林区东大街 239 号	029-87450287
PS2901	长安商场	雁塔区长安中路 38 号	029-85264950

(2) 每个商场包含有不同的部门，部门需要记录的信息包括部门编号(集团公司分配)、部门名称、位置分布和联系电话。某商场的部门信息如表 2-11 所示。

表 2-11　部门信息表

部门编号	部门名称	位置分布	联系电话
DT002	财务部	商场大楼 6 层	82504342
DT007	后勤部	商场地下 1 层	82504347
DT021	安保部	商场地下 1 层	82504358
DT005	人事部	商场大楼 6 层	82504446
DT004	管理部	商场裙楼 3 层	82504668

(3) 每个部门雇用多名员工处理日常事务，每名员工只能隶属于一个部门(新进员工在培训期不隶属于任何部门)。员工需要记录的信息包括员工编号(集团公司分配)、姓名、岗位、电话号码和工资。员工信息如表 2-12 所示。

(4) 每个部门的员工中有一名是经理，每个经理只能管理一个部门，系统需要记录每个经理的任职时间。

表 2-12　员工信息表

员工编号	姓　名	岗　位	电话号码	工资/元
XA3310	周超	理货员	13609257638	1500.00
SH1075	刘飞	防损员	13477293487	1500.00
XA0048	江雪花	广播员	15234567893	1428.00
BJ3123	张正华	部门主管	13345698432	1876.00

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-24 所示。

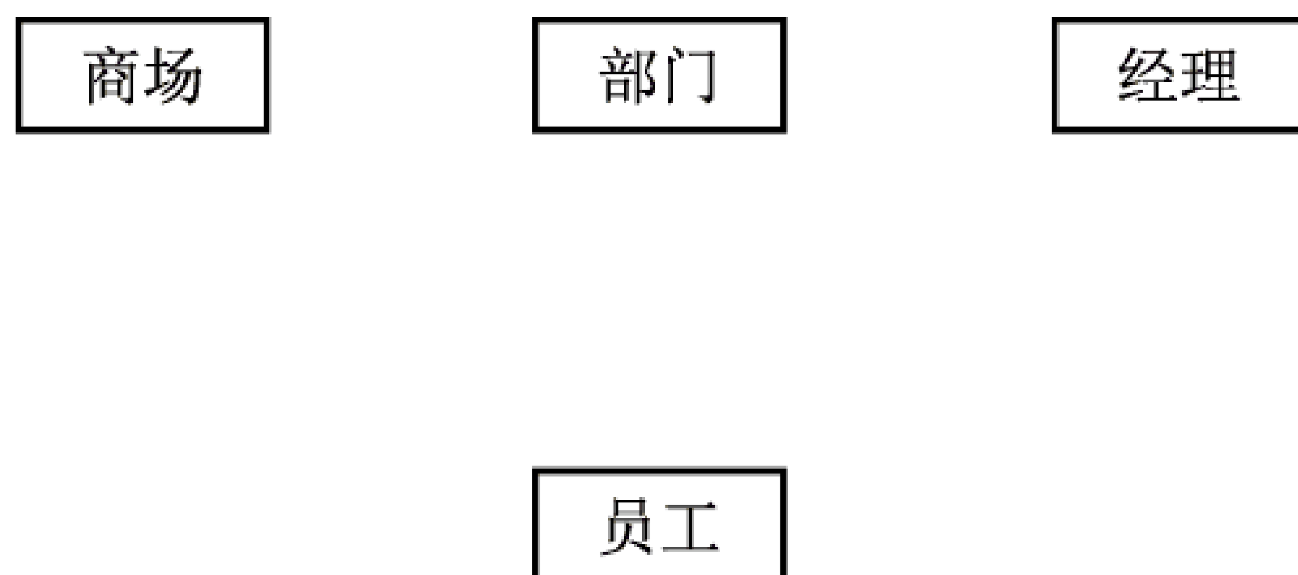


图 2-24　实体联系图

【关系模式设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式(不完整)：

商场(商场编号，商场名称，地址，联系电话)；

部门(部门编号，部门名称，位置分布，联系电话，__(a)__)；

员工(员工编号，员工姓名，岗位，电话号码，工资，__(b)__)；

经理(__(c)__，任职时间)。

【问题 1】

根据问题描述，补充 4 个联系，完善图 2-24 所示的实体联系图。联系名可用联系 1、联系 2、联系 3 和联系 4 代替，联系的类型分为 1∶1、1∶n 和 m∶n。

【问题 2】

根据实体联系图，将关系模式中的空(a)~(c)补充完整，并分别给出部门、员工和经理关系模式的主键和外键。

【问题 3】

为了使商场有紧急事务时能联系到轮休的员工，要求每位员工必须且只能登记一位紧急联系人的姓名和联系电话，不同的员工可以登记相同的紧急联系人，则在图 2-24 中还需添加的实体是__(1)__，该实体和图 2-24 中的员工存在__(2)__联系(填写联系类型)。给出该实体的关系模式。

9. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2008 年 12 月试题二)

【说明】

某宾馆拟开发一个宾馆客房预订子系统，主要是针对客房的预订和入住等情况进行管理。

【需求分析结果】

(1) 员工信息主要包括员工号、姓名、出生年月、性别、部门、岗位、住址、联系电话和密码等信息。岗位有管理和服务两种。岗位为“管理”的员工可以更改(添加、删除和修改)员工表中本部门员工的岗位和密码，要求将每一次更改前的信息保留；岗位为“服务”的员工只能修改员工表中本人的密码，且负责多个客房的清理等工作。

(2) 部门信息主要包括部门号、部门名称、部门负责人、电话等信息。一个员工只能属于一个部门，一个部门只有一位负责人。

(3) 客房信息包括客房号、类型、价格、状态等信息。其中，类型是指单人间、三人间、普通标准间、豪华标准间等；状态是指空闲、入住和维修。

(4) 客户信息包括身份证号、姓名、性别、单位和联系电话。

(5) 客房预订情况包括客房号、预订日期、预订入住日期、预订入住天数、身份证号等信息。一条预订信息必须且仅对应一位客户，但一位客户可以有多条预订信息。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-25 所示。

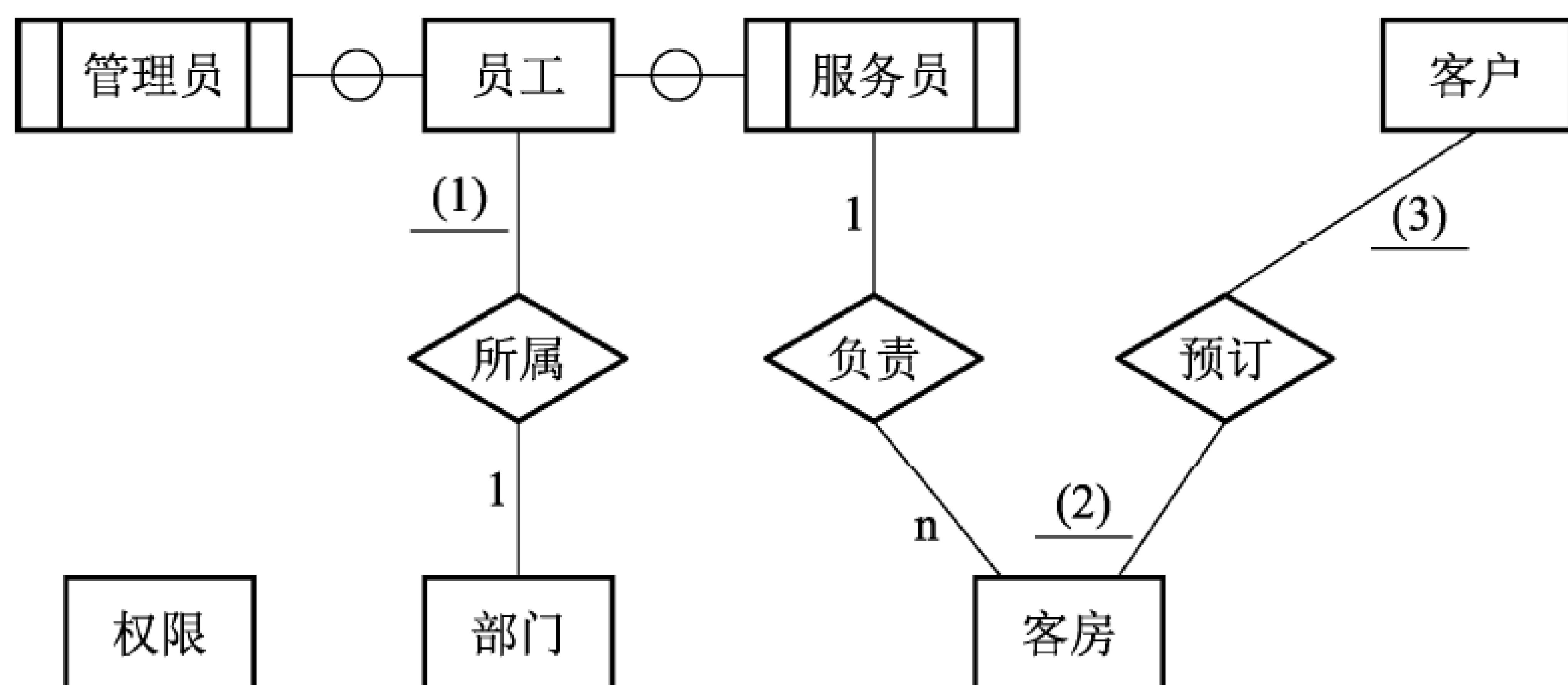


图 2-25　实体联系图

【逻辑结构设计】

逻辑结构设计阶段设计的部分关系模式(不完整)如下：

员工((4) ，姓名，出生年月，性别，岗位，住址，联系电话，密码)；

权限(岗位，操作权限)；

部门(部门号，部门名称，部门负责人，电话)；

客房((5) ，类型，价格，状态，入住日期，入住时间，员工号)；

客户((6) ，姓名，性别，单位，联系电话)；

更改权限(员工号， (7) ，密码，更改日期，更改时间，管理员号)；

预订情况((8) ，预订日期，预订入住日期，预订入住天数)。

【问题 1】

根据问题描述，填写图 2-25 中的(1)~(3)处联系的类型。联系类型分为一对一、一对多和多对多 3 种，分别使用 1∶1、1∶n 或 1∶*、m∶n 或*∶*表示。

【问题 2】

补充图 2-25 中的联系，并指明其联系类型。

【问题 3】

根据需求分析结果和图 2-25 所示，将逻辑结构设计阶段生成的关系模式中的空(4)~(8)补充完整。(注：一个空可能需要填多个属性。)

【问题 4】

若去掉权限表，并将权限表中的操作权限属性放在员工表中(仍保持管理和服务岗位的操作权限规定)，则与原有设计相比有什么优缺点(请从数据库设计的角度进行说明)?

10. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2008 年 5 月试题二)

【说明】

某地区举行篮球比赛，需要开发一个比赛信息管理系统来记录比赛的相关信息。

【需求分析结果】

(1) 登记参赛球队的信息。记录球队的名称、代表地区、成立时间等信息。系统记录球队每个队员的姓名、年龄、身高、体重等信息。每个球队有一个教练负责管理球队，一个教练仅负责一个球队。系统记录教练的姓名、年龄等信息。

(2) 安排球队的训练信息。比赛组织者为球队提供了若干个场地，供球队进行适应性训练。系统记录现有的场地信息，包括场地名称、场地规模、位置等信息。系统可为每个球队安排不同的训练场地，如表 2-13 所示，系统记录训练场地安排的信息。

表 2-13　训练场地安排表

球队名称	场地名称	训练时间
解放军	一号球场	2008-06-09　14:00~18:00
解放军	一号球场	2008-06-12　09:00~12:00
解放军	二号球场	2008-06-11　14:00~18:00
山西	一号球场	2008-06-10　09:00~12:00

(3) 安排比赛。该赛事聘请专职裁判，每场比赛只安排一个裁判。系统记录裁判的姓名、年龄、级别等信息。系统按照一定的规则，首先分组，然后根据球队、场地和裁判情况安排比赛(每场比赛的对阵双方分别称为甲队和乙队)。记录参赛球队名称、比赛时间、比分、比赛场地等信息，如表 2-14 所示。

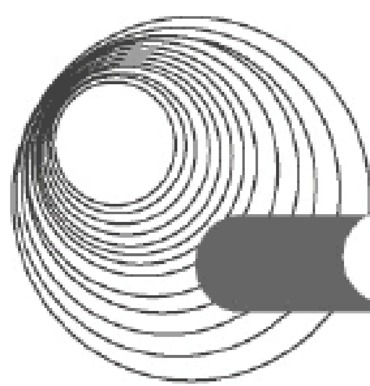
表 2-14　比赛安排表

A 组：

甲队—乙队	场地名称	比赛时间	裁　判	比　分
解放军—北京	一号球场	2008-06-17　15:00	李大明	
天津—山西	一号球场	2008-06-17　17:00	胡学海	

B 组：

甲队—乙队	场地名称	比赛时间	裁　判	比　分
上海—安徽	二号球场	2008-06-17　15:00	丁鸿平	
山东—辽宁	二号球场	2008-06-17　19:00	郭爱琪	



(4) 所有球员、教练和裁判可能在表中出现重名情况。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图和关系模式(不完整)如下。

(1) 实体联系图如图 2-26 所示。

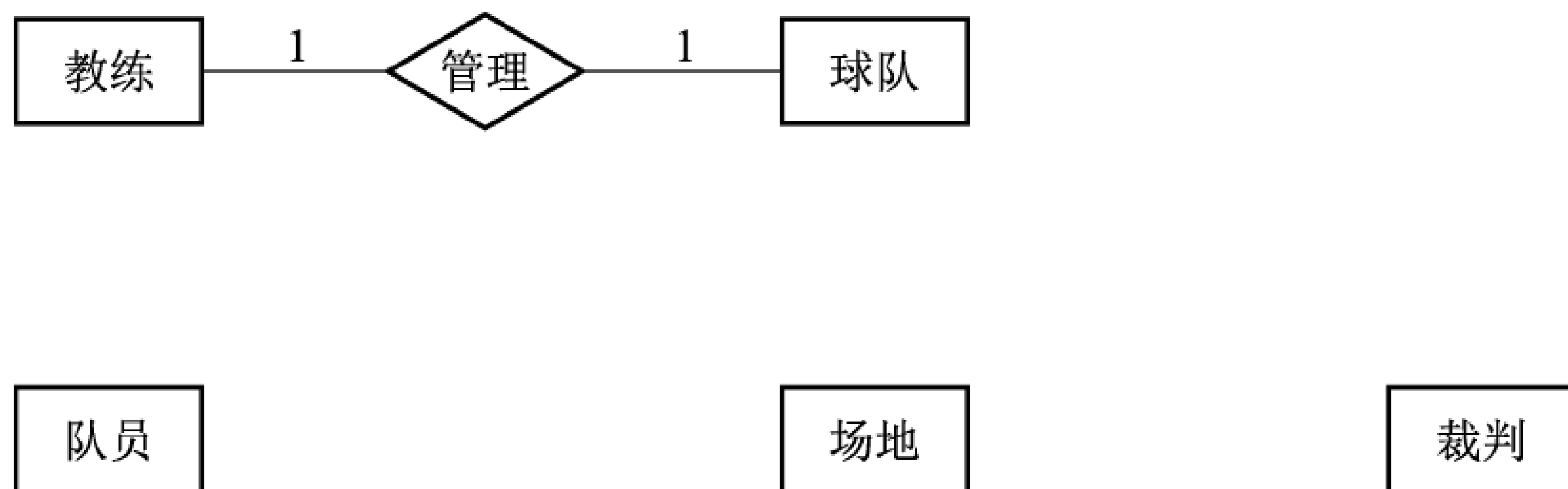


图 2-26　实体联系图

(2) 关系模式如下：

教练(教练编号，姓名，年龄)；

队员(队员编号，姓名，年龄，身高，体重， (a))；

球队(球队名称，代表地区，成立时间， (b))；

场地(场地名称，场地规模，位置)；

训练记录((c))；

裁判(裁判编号，姓名，年龄，级别)；

比赛记录((d))。

【问题 1】

根据问题描述，补充联系及其类型，完善图 2-26 所示的实体联系图(联系及其类型的书写格式参照教练与球队之间的联系描述，联系名称也可使用联系 1、联系 2 等)。

【问题 2】

根据实体联系图 2-26，填充关系模式中的(a)~(d)，并给出训练记录和比赛记录关系模式的主键和外键。

【问题 3】

如果考虑记录一些特别资深的热心球迷的情况，每个热心球迷可能支持多个球队。热心球迷信息包括姓名、住址和喜欢的俱乐部等基本信息。根据这一要求修改图 2-26 所示的实体联系图，给出修改后的关系模式(仅给出增加的关系模式描述)。

11. 阅读下列说明，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2007 年 11 月试题二)

【说明】

某汽车维修站拟开发一套小型汽车维修管理系统，对车辆的维修情况进行管理。

(1) 对于新客户及车辆，汽车维修管理系统首先登记客户信息，包括客户编号、客户名称、客户性质(个人、单位)、折扣率、联系人、联系电话等信息；还要记录客户的车辆信息，包括车牌号、车型、颜色等信息。一个客户至少有一台车。客户及车辆信息如表 2-15 所示。

(2) 记录维修车辆的故障信息，包括维修类型(普通、加急)、作业分类(大、中、小修)、

结算方式(自付、三包、索赔)等信息。维修厂的员工分为维修员和业务员。车辆维修首先委托给业务员。业务员对车辆进行检查和故障分析后，与客户磋商，确定故障现象，生成维修委托书，如表 2-16 所示。

表 2-15　客户及车辆信息

客户编号	GS0051	客户名称	××公司	客户性质	单位
折 扣 率	95%	联 系 人	杨浩东	联系电话	82638779
车 牌 号		**0765	车　型	帕萨特	
颜　色		白色	车辆类别	微型车	

表 2-16　维修委托书

No.20070702003　　　　登记日期：2007-07-02

车 牌 号	**0765	客户编号	GS0051	维修类型	普通
作业分类	中修	结算方式	自付	进厂时间	20070702 11:09
业 务 员	张小江	业务员编号	012	预计完工时间	
故障描述					
车头损坏，水箱漏水					

(3) 维修车间根据维修委托书和车辆的故障现象，在已有的维修项目中选择并确定一个或多个具体维修项目，安排相关的维修工及工时，生成维修派工单。维修派工单如表 2-17 所示。

表 2-17　维修派工单

No.20070702003

维修项目编号	维修项目	工　时	维修员编号	维修员工种
012	维修车头	5.00	012	机修
012	维修车头	2.00	023	漆工
015	水箱焊接补漏	1.00	006	焊工
017	更换车灯	1.00	012	机修

(4) 客户车辆在车间修理完毕后，根据维修项目单价和维修派工单中的工时计算车辆此次维修的总费用，并记录在委托书中。

【概念结构设计】

根据需求阶段收集的信息，设计的实体联系图(不完整)如图 2-27 所示。图 2-27 的业务员和维修工是员工的子实体。

【逻辑结构设计】

逻辑结构设计阶段设计的关系模式(不完整)如下：

客户((5) ，折扣率，联系人，联系电话)；

车辆(车牌号，客户编号，车型，颜色，车辆类别)；

委托书((6) ，维修类型，作业分类，结算方式，进厂时间，预计完工时间，登记日期，

故障描述，总费用)；

维修项目(维修项目编号，维修项目，单价)；

派工单((7) ，工时)；

员工((8) ，工种，员工类型，级别)。

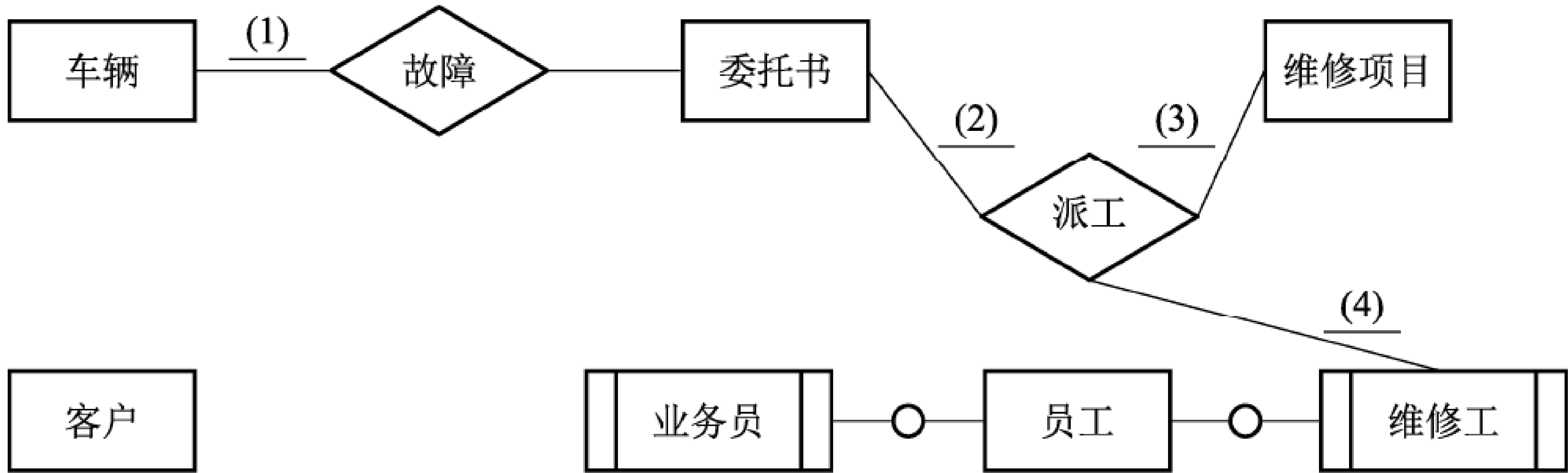


图 2-27　实体联系图

【问题 1】

根据问题描述，填写图 2-27 中(1)~(4)处联系的类型。联系类型分为一对一、一对多和多对多 3 种，分别使用 1∶1、1∶n 或 1∶*、m∶n 或*∶*表示。

【问题 2】

补充图 2-27 所示的联系并指明其联系类型。联系名可为联系 1、联系 2 等。

【问题 3】

根据图 2-27 和说明，将逻辑结构设计阶段生成的关系模式中的空(5)~(8)补充完整。

【问题 4】

根据问题描述，写出客户、委托书和派工单这 3 个关系模式的主键。

12. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2007 年 5 月试题二)

【说明】

某医院的门诊管理系统实现了为患者提供挂号、处方药品收费的功能。具体的需求及设计如下。

(1) 医院医师具有编号、姓名、科室、职称、出诊类型和出诊费用，其中出诊类型分为专家门诊和普通门诊，与医师职称无关；各个医师可以具有不同的出诊费用，与职称和出诊类型无关。

(2) 患者首先在门诊挂号处挂号，选择科室和医师，根据选择的医师缴纳挂号费(医师出诊费)。收银员为患者生成挂号单，如表 2-18 所示，其中，就诊类型为医师的出诊类型。

表 2-18　××医院门诊挂号单

收银员：13011　　　　时间：2007 年 2 月 1 日 08:58

就 诊 号	姓　名	科　室	医　师	就诊类型	挂 号 费
20070205015	叶萌	内科	杨玉明	专家门诊	5 元

(3) 患者在医师处就诊后，凭借挂号单和医师手写处方到门诊药房交费买药。收银员根

据就诊号和医师处方中开列的药品信息，查询药品库(如表 2-19 所示)，生成门诊处方单(如表 2-20 所示)。

表 2-19　药品库

药品编码	药品名称	类　型	库　存	货架编号	单　位	规　格	单价/元
12007	牛蒡子	中药	51590	B1410	G	炒	0.0340
11090	百部	中药	36950	B1523	G	片	0.0313

表 2-20　××医院门诊处方单

时间：2007 年 2 月 1 日　10:31

就 诊 号	20070205015	病人名称	叶萌	医师姓名	杨玉明
金额总计	0.65	项目总计	2	收 银 员	21081
药品编码	药品名称	数　量	单　位	单价/元	金额/元
12007	牛蒡子	10	G	0.0340	0.34
11090	百部	10	G	0.0313	0.31

(4) 由于药品价格会发生变化，因此门诊管理系统必须记录处方单上药品的单价。

根据需求阶段收集的信息，设计的实体联系图和关系模式(不完整)如下。

【实体联系图】(见图 2-28)

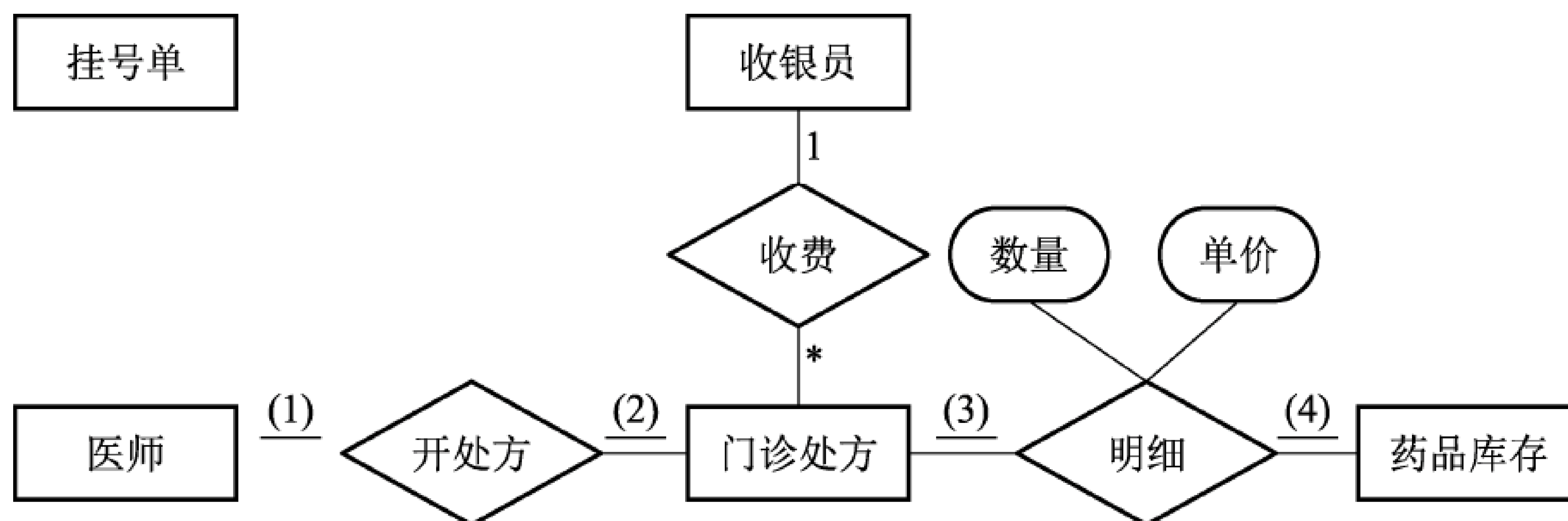


图 2-28　实体联系图

【关系模式】

挂号单(就诊号，病患姓名，医师编号，时间，__(5)__)；

收银员(编号，姓名，级别)；

医师(编号，姓名，科室，职称，出诊类型，出诊费用)；

门诊处方(__(6)__，收银员，时间)；

处方明细(就诊号，__(7)__)；

药品库(药品编码，药品名称，__(8)__)。

【问题 1】

根据问题描述，填写图 2-28 所示的实体联系图中(1)~(4)处联系的类型。

【问题 2】

图 2-28 中还缺少几个联系？请指出每个联系两端的实体名，格式如下：

实体 1∶实体 2

例如，若收银员与门诊处方之间存在联系，则表示如下：

收银员∶门诊处方　或　门诊处方∶收银员

【问题 3】

根据实体联系图 2-28 填写挂号单、门诊处方、处方明细和药品库关系模式中的空(5)~(8)处，并指出挂号单、门诊处方和处方明细关系模式的主键。

13. 阅读以下说明，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2006 年 11 月试题二)

【说明】

某宾馆需要建立一个住房管理系统，部分需求分析结果如下。

(1) 一个房间有多个床位，同一房间内的床位具有相同的收费标准。不同房间的床位收费标准可能不同。

(2) 每个房间有房间号(如 201、202 等)、收费标准、床位数目等信息。

(3) 每位客人有身份证号码、姓名、性别、出生日期和地址等信息。

(4) 对每位客人的每次住宿，应该记录其入住日期、退房日期和预付款额等信息。

(5) 管理系统可查询出客人所住房间号。

根据以上需求分析结果，设计一种关系模型实体联系图，如图 2-29 所示。



图 2-29　住房管理系统实体联系图

【问题 1】

根据上述说明和实体联系图，得到该住房管理系统的关系模式如下所示，请补充住宿关系。

房间(房间号，收费标准，床位数目)；

客人(身份证号，姓名，性别，出生日期，地址)；

住宿((1) ，入住日期，退房日期，预付款额)。

【问题 2】

请给出问题 1 中住宿关系模式的主键和外键。

【问题 3】

若将上述各关系直接实现为对应的物理表，现需查询在 2005 年 1 月 1 日至 2005 年 12 月 31 日期间，在该宾馆住宿次数大于 5 次的客人身份证号，并且按照入住次数进行降序排列。下面是实现该功能的 SQL 语句，请填补语句中的空缺。

```
SELECT 住宿.身份证号,count(入住日期) FROM 住宿,客人
WHERE 入住日期 >='20050101' AND 入住日期 <='20051231'
AND 住宿.身份证号 = 客人.身份证号
GROUP BY  (2)
 (3)  count(入住日期) > 5
 (4)
```

【问题 4】

为加快 SQL 语句的执行效率，可在相应的表上创建索引。根据问题 3 中的 SQL 语句，除主键和外键外，还需要在哪个表的哪些属性上创建索引？应该创建什么类型的索引？请说明原因。

14. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2006 年 5 月试题三)

【说明】

某单位资料室需要建立一个图书管理系统，初步的需求分析结果如下。

(1) 资料室有图书管理员若干名，他们负责已购入图书的编目和借还工作，每名图书管理员的信息包括工号和姓名。

(2) 读者可在阅览室读书，也可通过图书流通室借还图书，读者信息包括读者 ID、姓名、电话和 E-mail，系统为不同读者生成不同的读者 ID。

(3) 每部书在系统中对应唯一的一条图书在版编目数据(CIP，以下简称书目)，书目的基本信息包括 ISBN 号、书名、作者、出版商、出版年月，以及本资料室拥有该书的册数(以下简称册数)，不同书目的 ISBN 号不同。

(4) 资料室对于同一书目的图书可拥有多册(本)，图书信息包括图书 ID、ISBN 号、存放位置、当前状态，每一本书在系统中被赋予唯一的图书 ID。

(5) 一名读者最多只能借阅 10 本图书，且每本图书最多只能借两个月，读者借书时需由图书管理员登记读者 ID、所借图书 ID、借阅时间和应还时间，读者还书时图书管理员在对应的借书信息中记录归还时间。

(6) 当某书目的可借出图书的数量为零时，读者可以对其进行预约登记，即记录读者 ID、需要借阅图书的 ISBN 号、预约时间等。

某书目的信息如表 2-21 所示，与该书目对应的图书信息如表 2-22 所示。

表 2-21　书目信息

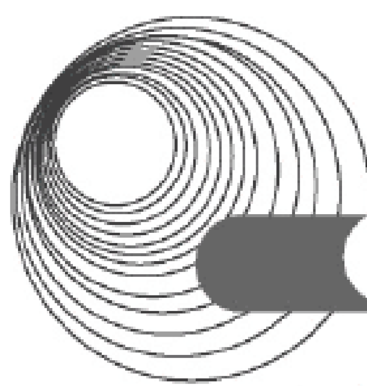
书　名	作　者	出 版 商	ISBN 号	出版年月	册　数	经 办 人
数据结构	严蔚敏 吴伟民	清华大学出版社	ISBN 978-7-302-02368-9	1997.4	4	01

表 2-22　图书信息

图书 ID	ISBN 号	存放位置	状　态	经 办 人
C832.1	ISBN 978-7-302-02368-9	图书流通室	已借出	01
C832.2	ISBN 978-7-302-02368-9	图书阅览室	不外借	01
C832.3	ISBN 978-7-302-02368-9	图书流通室	未借出	01
C832.4	ISBN 978-7-302-02368-9	图书流通室	已预约	01

系统的主要业务处理如下。

(1) 入库管理。图书购进入库时，管理员查询本资料室的书目信息，若该书的书目尚未



建立，则由管理员编写该书的书目信息并录入系统；否则修改该书目的册数，然后编写并录入图书信息。对于进入流通室的书，其初始状态为“未借出”；而送入阅览室的书，其状态始终为“不外借”。

(2) 借书管理。读者借书时，若有，则由管理员为该读者办理借书手续，并记录该读者的借书信息，同时将借出图书的状态修改为“已借出”。

(3) 预约管理。若图书流通室没有读者要借的书，则可为该读者建立预约登记，需要记录读者 ID、书的 ISBN 号、预约时间和预约期限(最长为 10 天)。一旦其他读者归还这种书，就自动通知该预约读者，系统将自动清除超出预约期限的预约记录并修改相关信息。

(4) 还书管理。读者还书时，则记录相应借还信息中的“归还时间”，对于超期归还者，系统自动计算罚金(具体的计算过程此处省略)。系统同时自动查询预约登记表，若存在其他读者预约该书的记录，则将该图书的状态修改为“已预约”，并将该图书 ID 写入相应的预约记录中(系统在清除超出预约期限的记录时解除该图书的“已预约”状态)；否则，将该图书的状态修改为“未借出”。

(5) 通知处理。对于已到期且未归还的图书，系统通过 E-mail 自动通知读者；若读者预约的书已到，系统则自动通过 E-mail 通知该读者来办理借书手续。

【问题 1】

根据以上说明设计的实体联系图如图 2-30 所示，请指出读者与图书、书目与读者、书目与图书之间的联系类型。

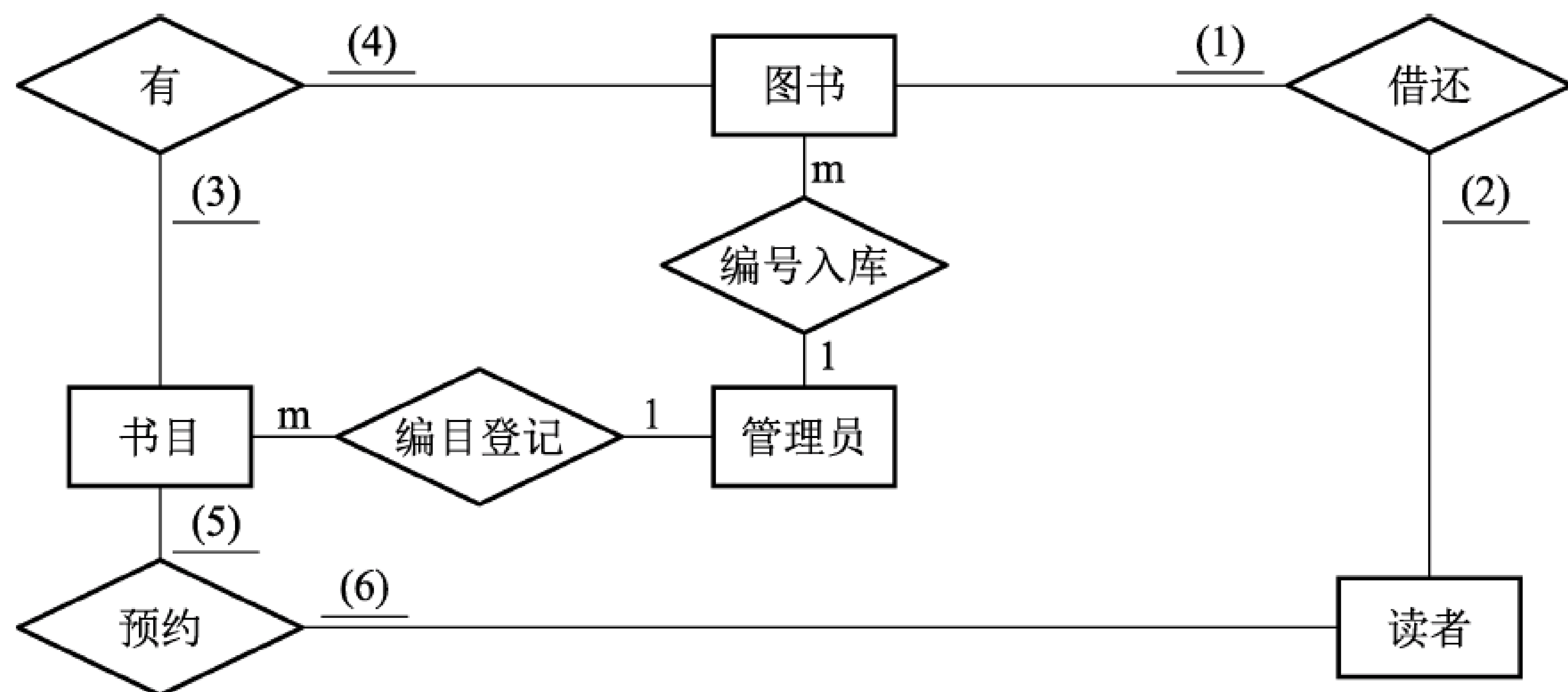


图 2-30　图书管理系统的实体联系图

【问题 2】

该图书管理系统的主要关系模式如下，请补充借还记录关系模式和预约登记关系模式中的空缺。

管理员(工号，姓名)；

读者(读者 ID，姓名，电话，E-mail)；

书目(ISBN 号，书名，作者，出版商，出版年月，册数，经办人)；

图书(图书 ID，ISBN 号，存放位置，状态，经办人)；

借还记录(<u>(a)</u>，借出时间，应还时间，归还时间)；

预约登记(<u>(b)</u>，预约时间，预约期限，图书 ID)；

注：时间格式为“年.月.日 时:分:秒”。

【问题 3】

请指出问题 2 中给出的读者、书目关系模式的主键，以及图书、借还记录和预约登记关系模式的主键和外键。

2.1.4　同步练习参考答案

1.

【问题 1】

完善后的实体联系图如图 2-31 所示。

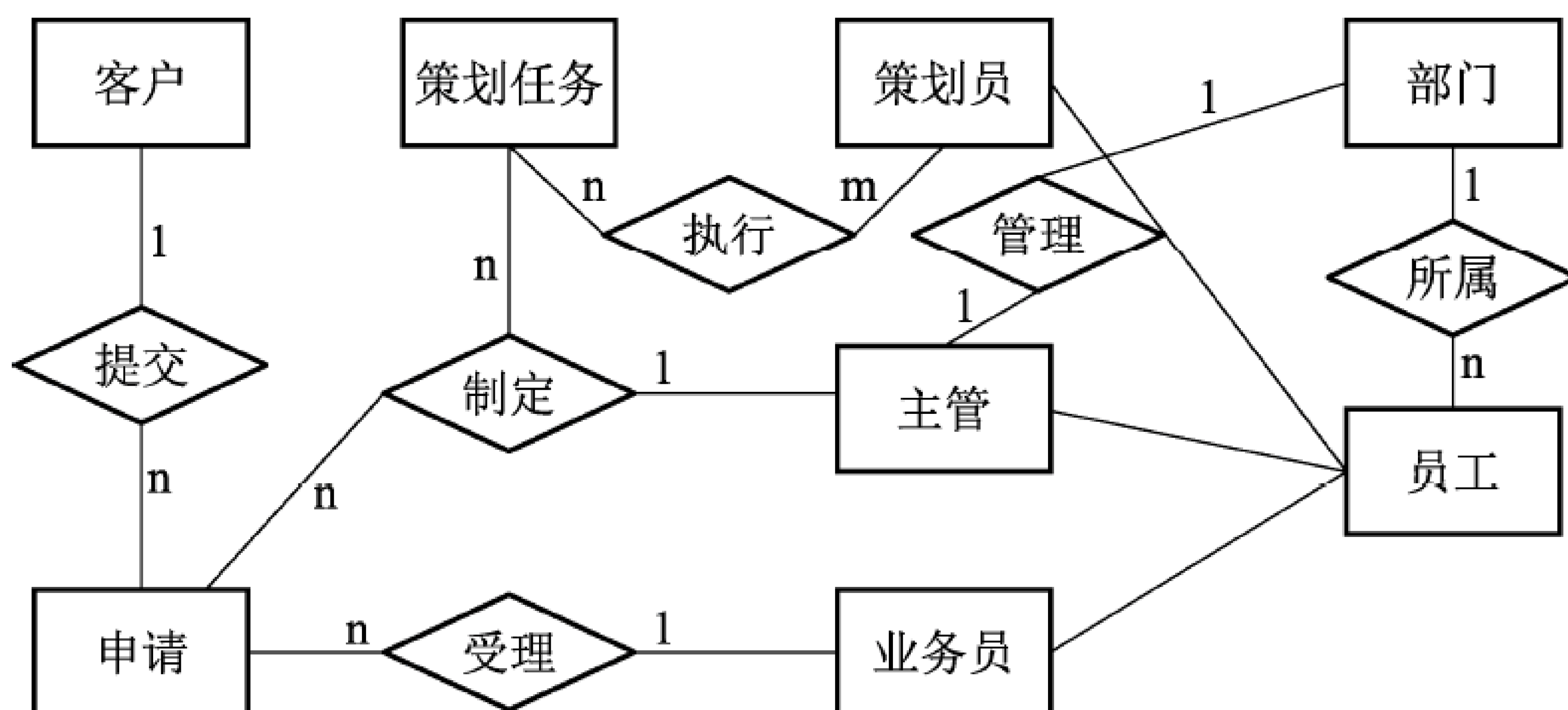


图 2-31　完善后的实体联系图

【问题 2】

(a) 部门号，职位。

(b) 申请号，客房类型，客户号。其中主键为申请号，外键为(客户号#、员工号#)。

(c) 申请号，员工号。其中主键和外键均为申请号和员工号。

(d) 申请号，员工号。其中主键为(申请号、员工号、实际完成时间)，外键为申请号#、员工号#。

【问题 3】

会议申请关系模式存在数据冗余及数据修改的不一致性问题，应将其分解为如下两个关系模式：会议申请 1(申请号，客户号，开会日期，会议地点，持续天数，会议人数，预算费用，会议类型，酒店要求，会议室要求，联系人，联系方式，受理标志，员工号)及会议申请 2(申请号，客房类型，客房数)。

2.

【问题 1】

补充后的实体联系图如图 2-32 所示。

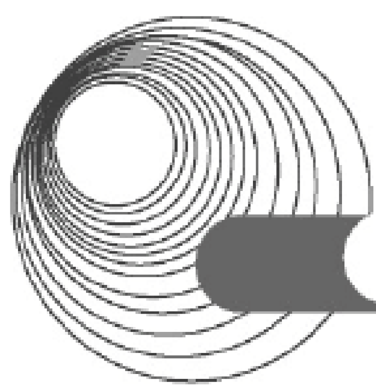
【问题 2】

(1) 病区，护士编号。

(2) 手术室号，护士编号，责任。

(3) 病案号，病床号。

(4) 病案号，医生编号。



(5) 病案号，手术室号，手术时间，医生编号。

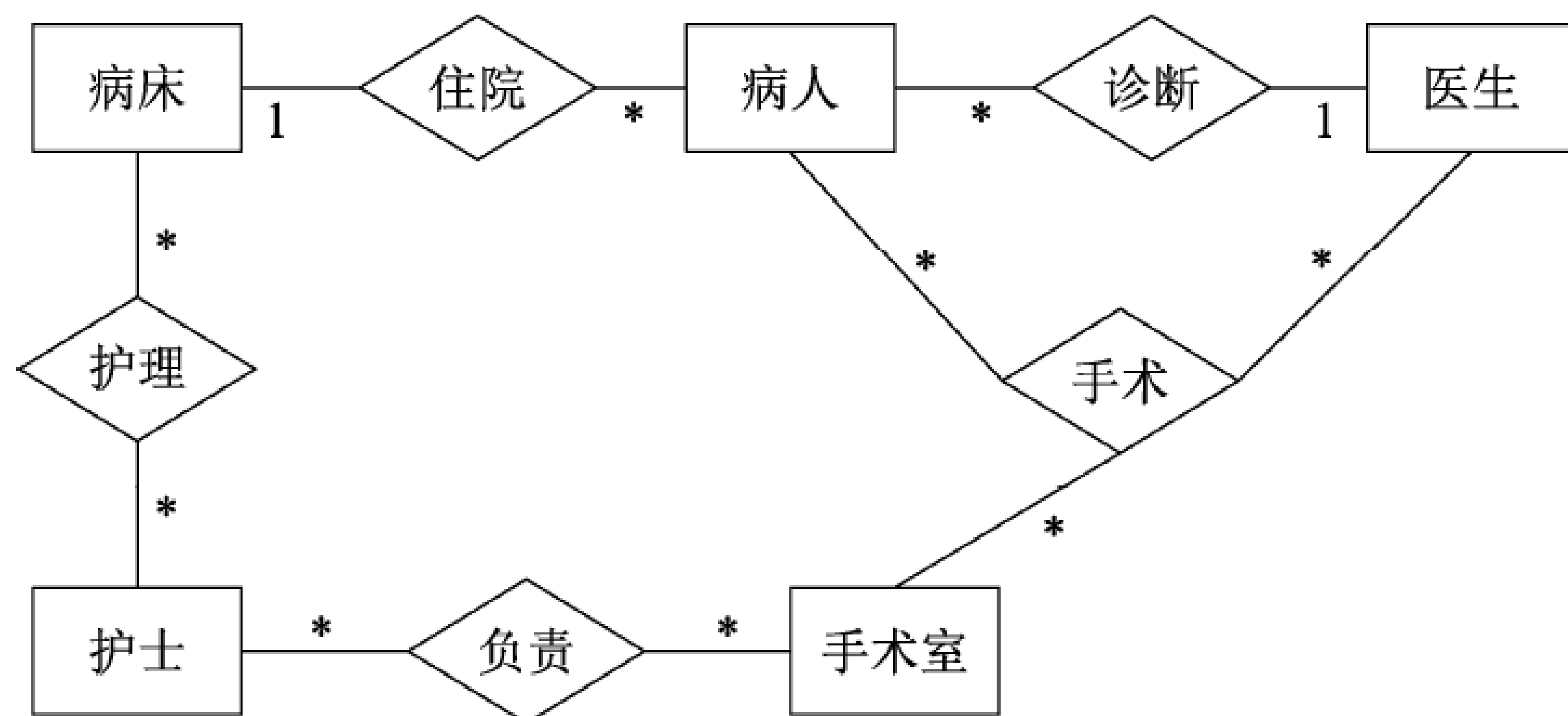


图 2-32　补充后的实体联系图

【问题 3】

补充用药情况后的实体联系图如图 2-33 所示。

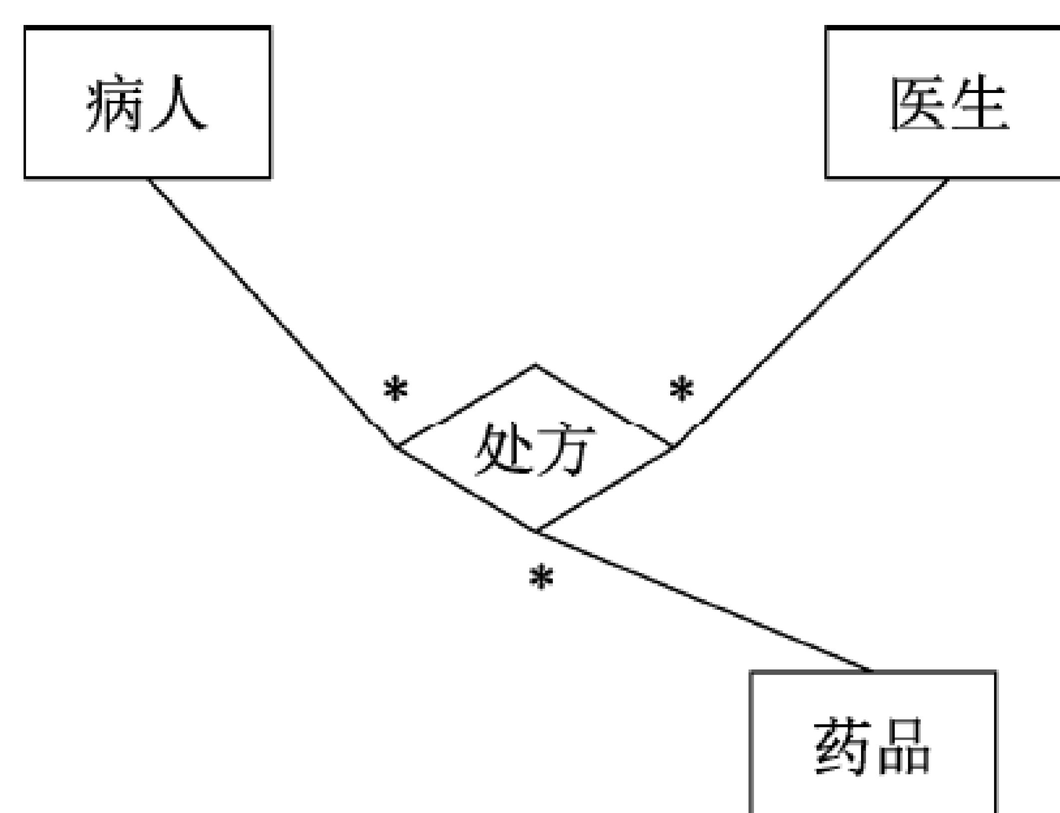


图 2-33　补充用药情况后的实体联系图

3.

【问题 1】

完善后的实体联系图如图 2-34 所示。

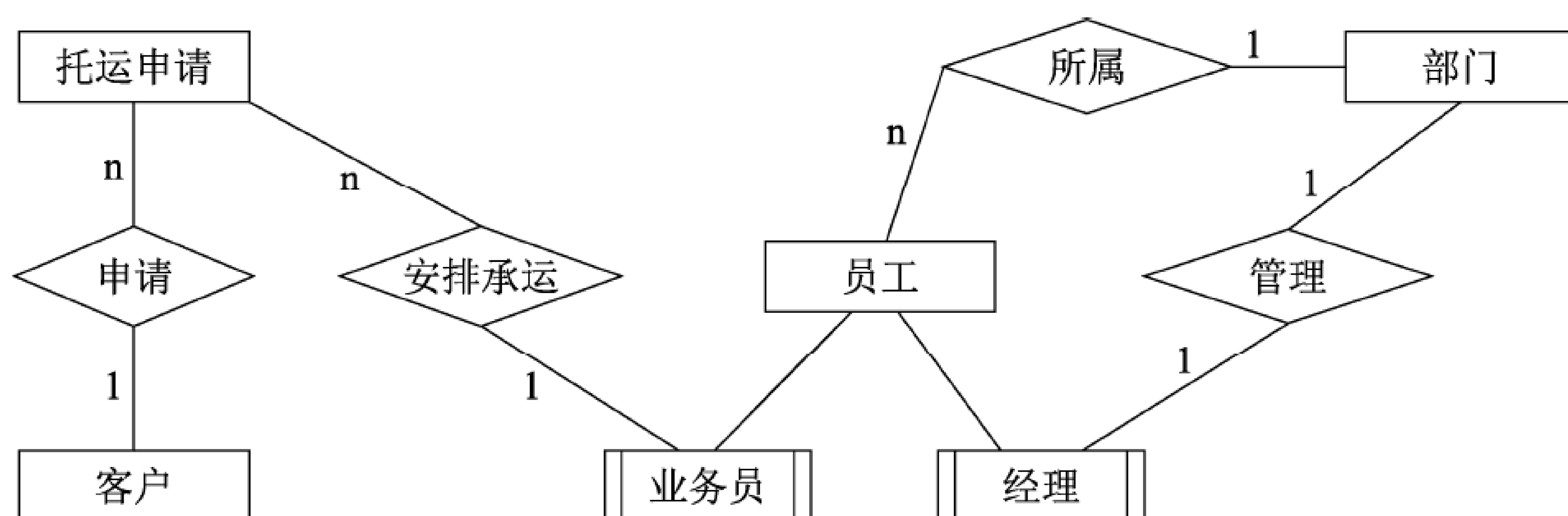


图 2-34　完善后的实体联系图

【问题 2】

(a) 部门号；(b) 客户号；(c) 申请号，客户号；(d) 申请号。

部门关系的主键：部门号；外键：经理。

员工关系的主键：员工号；外键：部门号。

安排承运关系的主键：申请号；外键：业务员。

【问题 3】

修改后的实体联系图如图 2-35 所示。

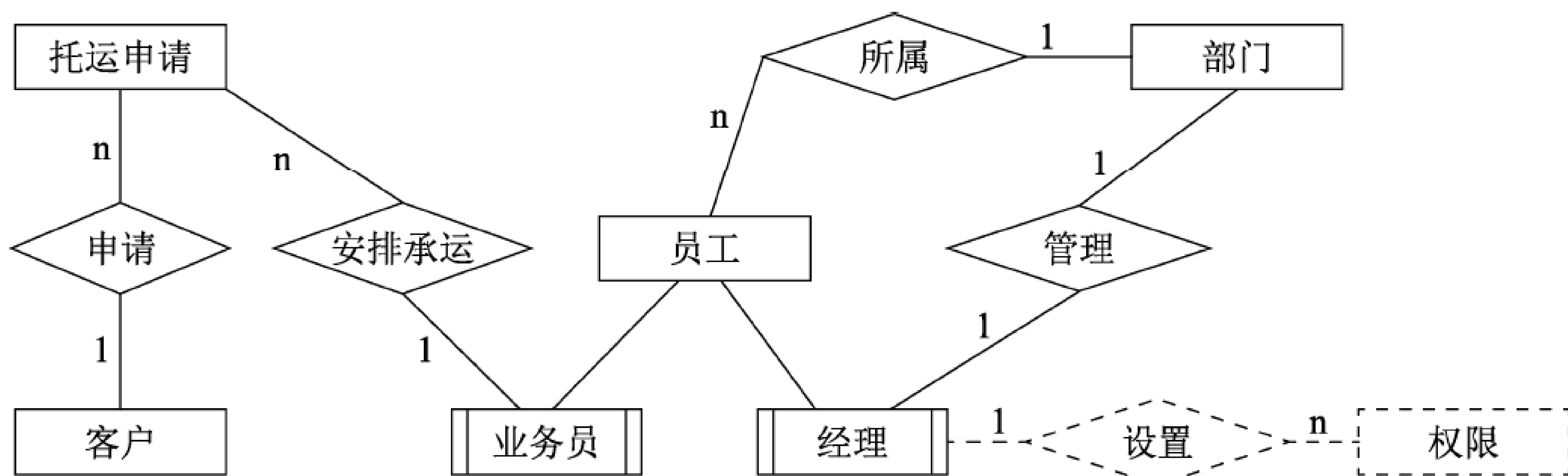


图 2-35　修改后的实体联系图

增加的关系模式为：

权限(员工号，权限，设置人)或权限(员工号，权限，部门经理)。

4.

【问题 1】

补充后的实体联系图如图 2-36 所示。

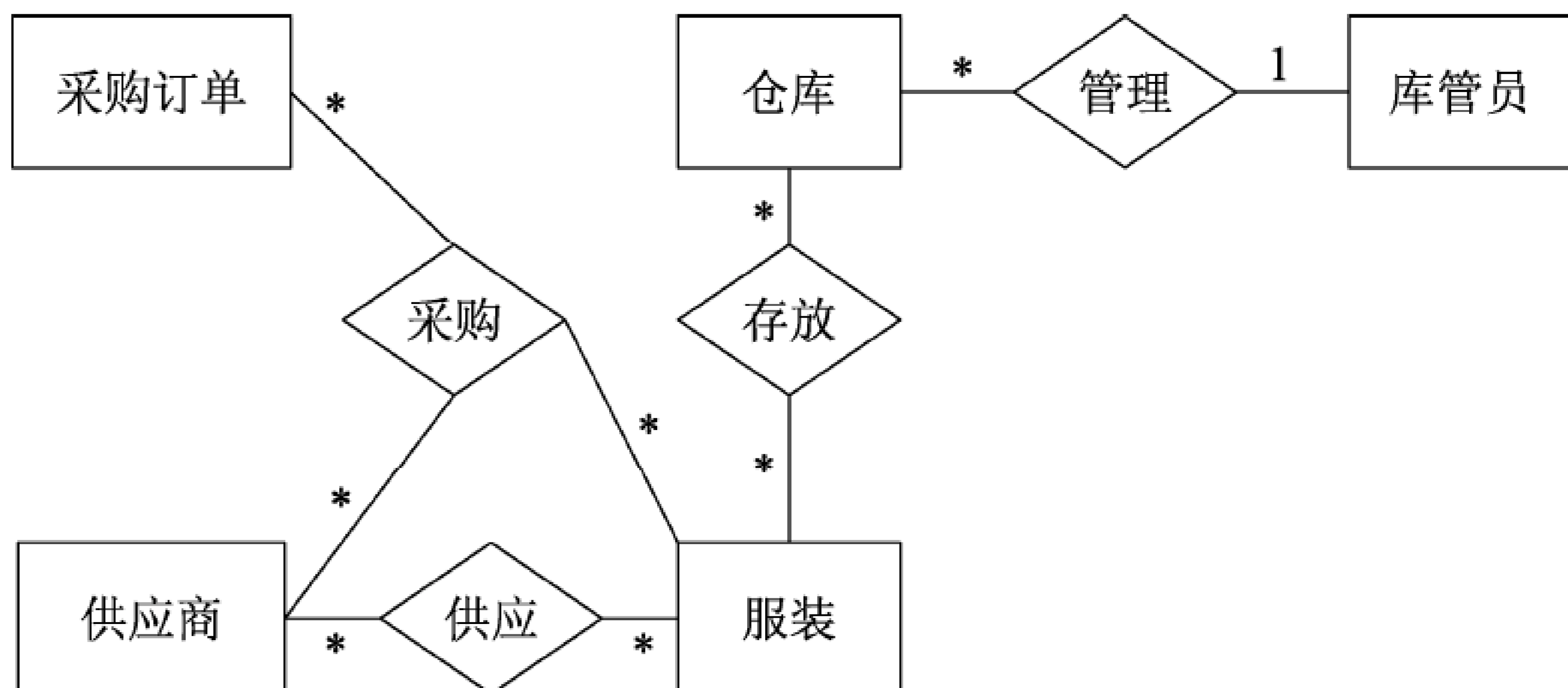


图 2-36　补充后的实体联系图

【问题 2】

(1) 仓库编号，库管员编码；

(2) 供应商编码，服装编码；

(3) 订单编码，订货日期，应到货日期；

(4) 订单编码，服装编码，供应商编码，数量，采购价格。

【问题 3】

修改后的实体联系图如图 2-37 所示。

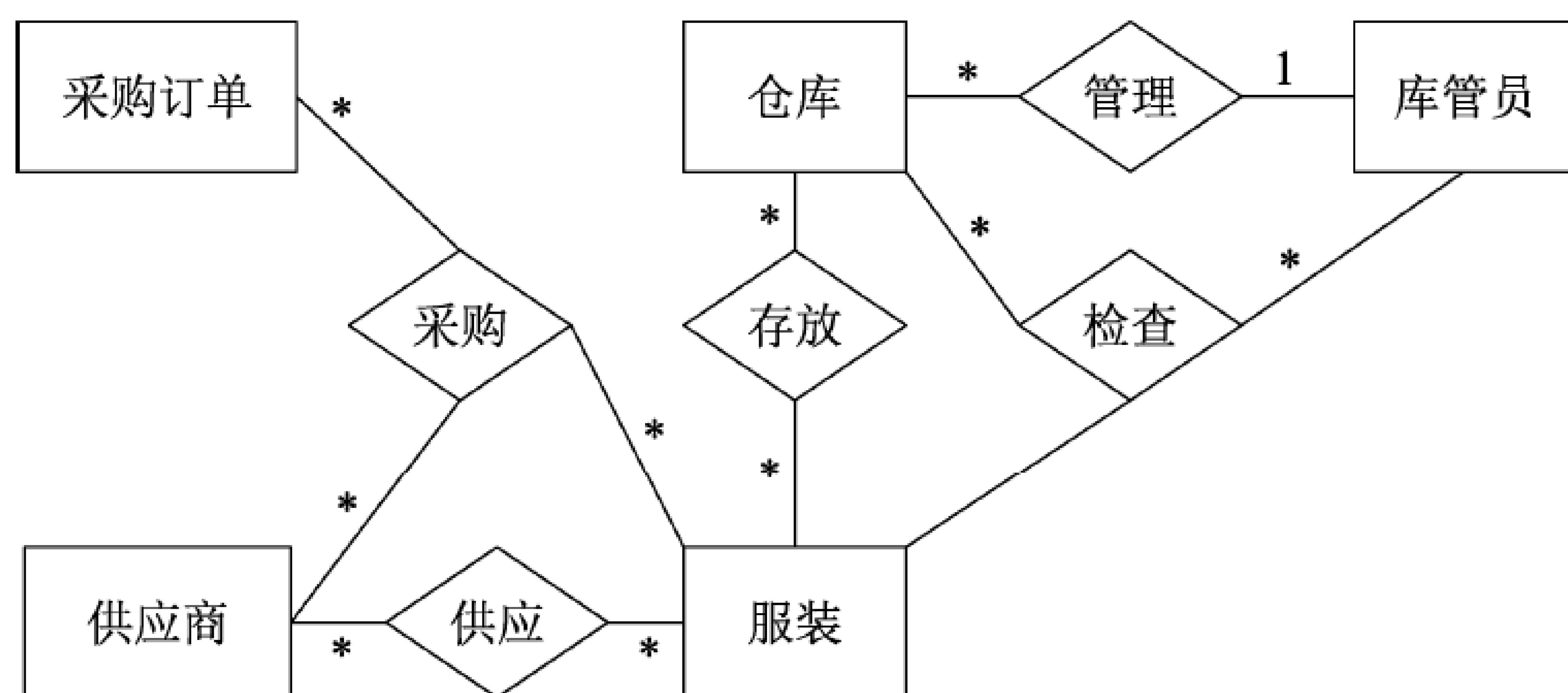


图 2-37 修改后的实体联系图

5.

【问题 1】

(1) 房号，业主编号。业主关系模式的主键：房号；外键：无。

(2) 员工号，部门号。员工关系模式的主键：员工号；外键：部门号。

(3) 部门号，部门负责人。部门关系模式的主键：部门号；外键：部门负责人。

(4) 收费类型，单位，单价。收费标准关系模式的主键：收费类型；外键：无。

(5) 房号，业主编号，收费日期。收费信息关系模式的主键：房号，业主编号，收费日期；外键：房号，收费类型，员工号。

【问题 2】

(a) m；(b) n；(c) 1；(d) *；(e) 1；(f) *。

添加一个实体——收费标准，与“收费”连接，类型为*，如图 2-38 所示。

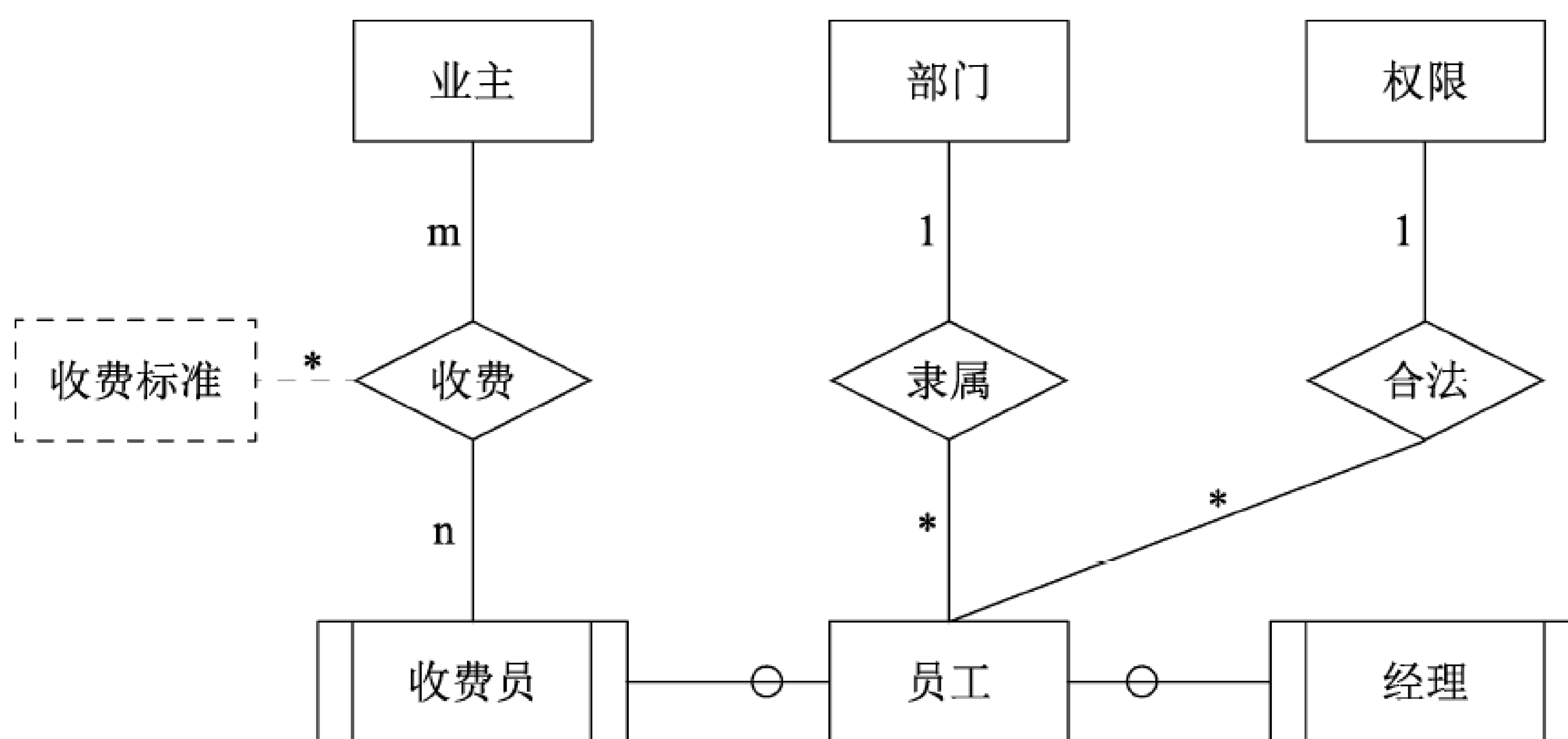


图 2-38 补充后的实体联系图

【问题 3】

业主关系是第二范式。

存在的问题：数据冗余，当一个业主有多套房时，重复存储多份姓名、工作单位、联系电话。

6.

【问题 1】

补充后的实体联系图如图 2-39 所示。

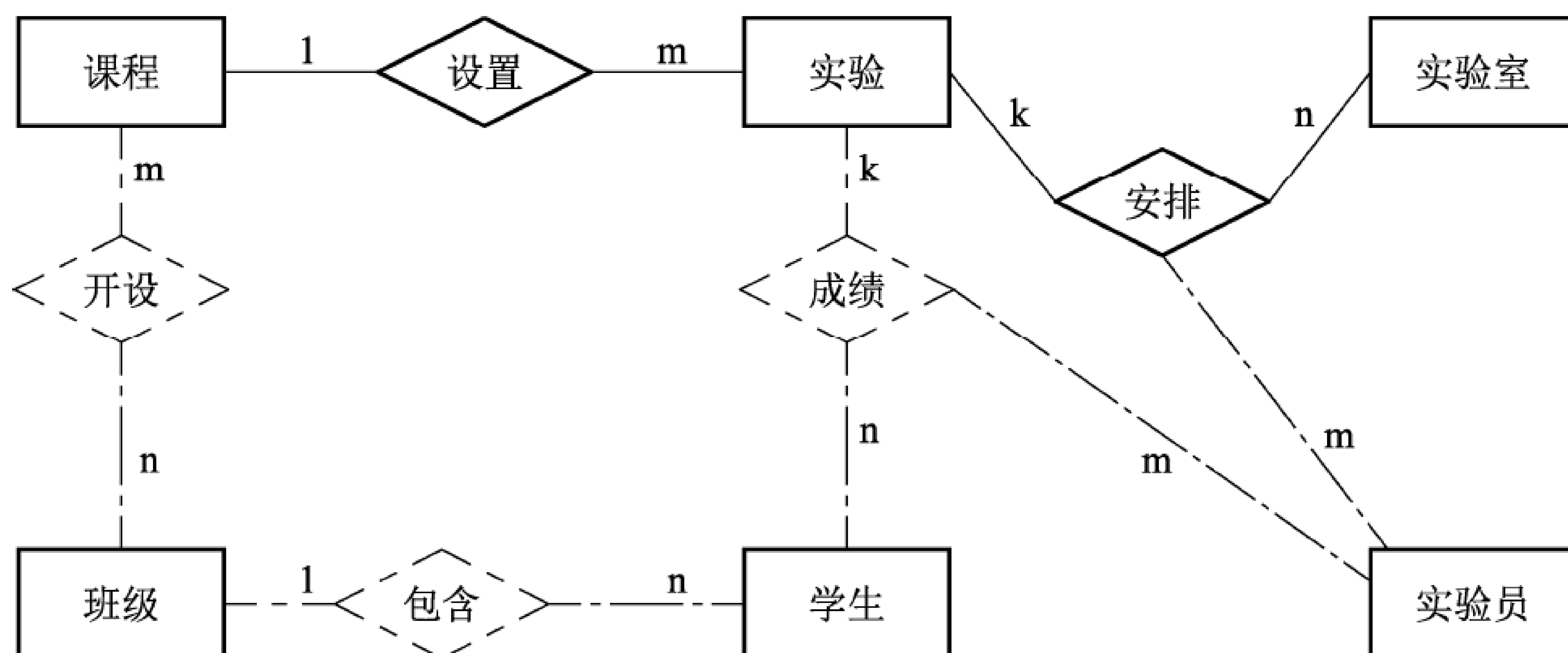


图 2-39　补充后的实体联系图

【问题 2】

(1) 课程编号，班级号；

(2) 实验编号，课程编号；

(3) 实验编号，批次号，安排学期，实验室编号，实验员编号；

(4) 实验员编号，实验员姓名；

(5) 学号，班级号；

(6) 实验编号，学号。

【问题 3】

修改后的实体联系图如图 2-40 所示。

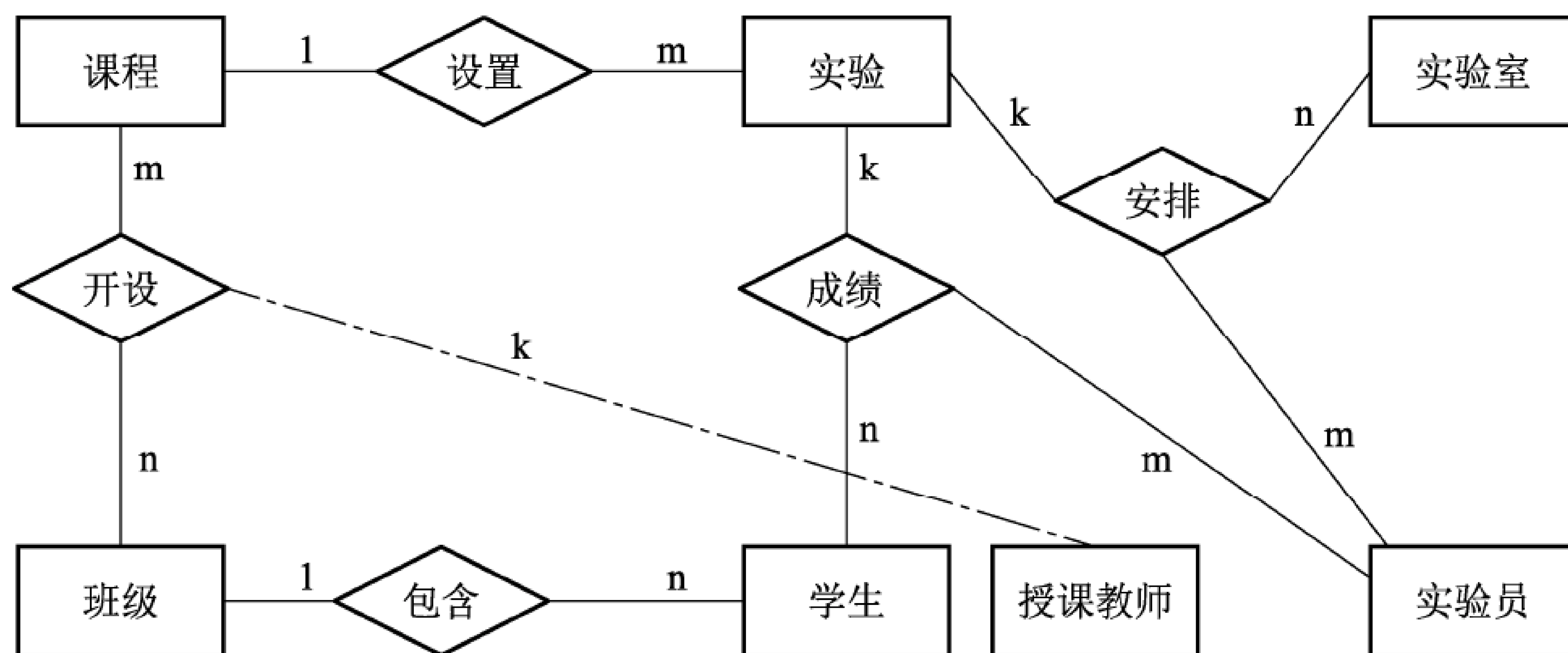


图 2-40　修改后的实体联系图

7.

【问题 1】

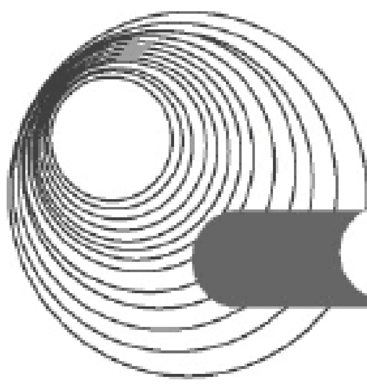
地址簿与用户：一对一；

邮件账号与邮件：一对多；

邮件与附件：一对多。

【问题 2】

(a) 用户名；(b) 邮件号，发件人地址；(c) 邮件号。



【问题 3】

(1) 地址簿、邮件和附件关系模式的主键和外键如表 2-23 所示。

表 2-23　地址簿、邮件和附件关系模式的主键和外键

关系模式	主　键	外　键
地址簿	联系人编号	用户名
邮件	邮件号	发件人地址或收件人地址
附件	邮件号，附件号	邮件号

(2) 附件属于弱实体。一个实体的键是由另一个实体的部分或全部属性构成，这样的实体称为弱实体。附件的外键邮件号是属于邮件这个实体的，所以它属于弱实体，依赖于邮件这个实体。

8.

【问题 1】

完整的实体联系图如图 2-41 所示。

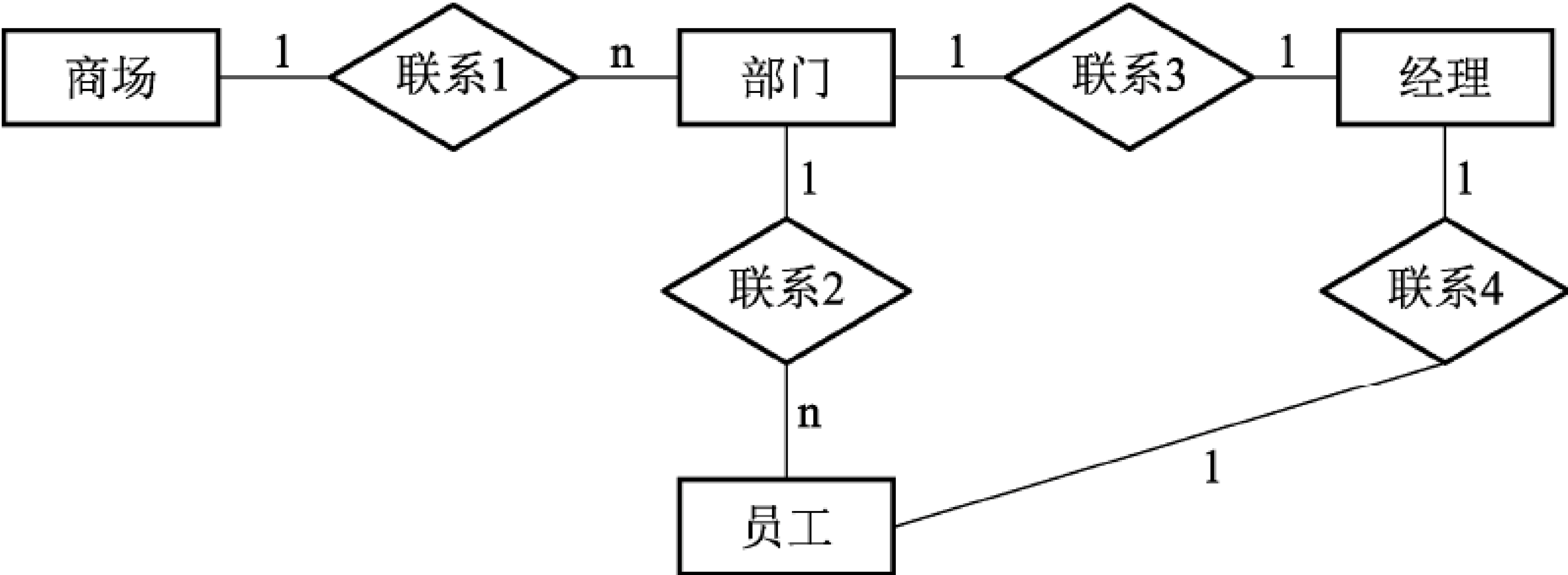


图 2-41　完整的实体联系图

【问题 2】

(a) 商场编号；(b) 部门编号；(c) 员工编号。

部门、员工和经理关系模式的主键和外键如表 2-24 所示。

表 2-24　部门、员工、经理关系模式的主键和外键

关系模式	主　键	外　键
部门	部门编号	商场编号
员工	员工编号	部门编号
经理	员工编号	员工编号

【问题 3】

(1) 紧急联系人。

(2) 1∶n。

紧急联系人(员工编号，姓名，联系电话)。

主键：员工编号。

9.

【问题 1】

(1) n 或 m 或*；(2) n 或 m 或*；(3) n 或 m 或*。

【问题 2】

员工到权限的联系，联系类型为 m∶1。

【问题 3】

(4) 员工号，部门号；(5) 客房号；(6) 身份证号；(7) 岗位；(8) 客房号，身份证号。

【问题 4】

优点：如果合为一个表，则只查一次表就能得出岗位和操作权限信息，加快了查找速度。缺点：如果合为一个表，则岗位、操作权限将多次重复出现，会产生冗余数据并增加数据库存储量。

10.

【问题 1】

完善后的实体联系图如图 2-42 所示。

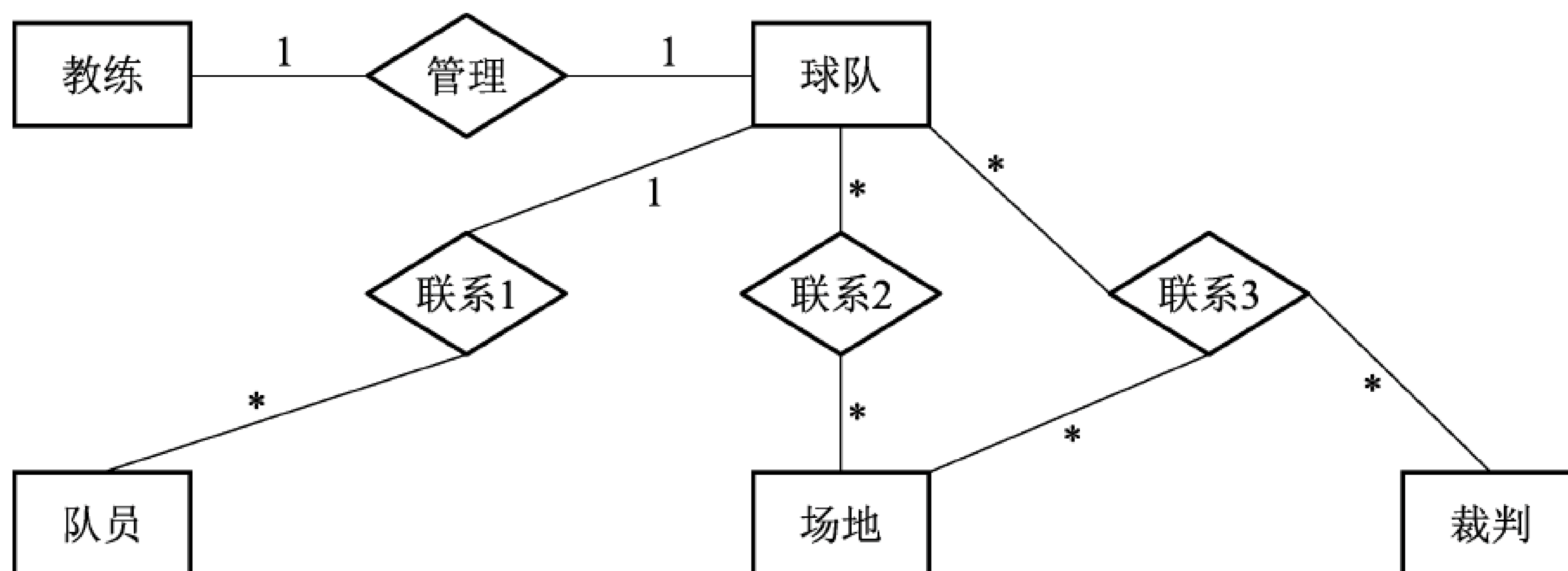


图 2-42　完善后的实体联系图

【问题 2】

(a) 球队名称；

(b) 教练编号；

(c) 球队名称，场地名称，开始时间，结束时间；

(d) 甲队，乙队，比赛时间，球场名称，比分，裁判，分组。

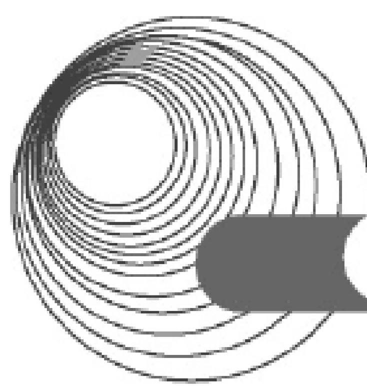
训练记录和比赛记录关系模式的主键和外键如表 2-25 所示。

表 2-25　训练记录和比赛记录关系模式的主键和外键

训练记录	主键	球队，开始时间；或球队，结束时间；或场地名称，开始时间；或场地名称，结束时间
	外键	球队名称，场地名称
比赛记录	主键	甲队，比赛时间；或场地名称，比赛时间；或裁判，比赛时间；或乙队，比赛时间
	外键	甲队，乙队，场地名称，裁判

【问题 3】

修改后的实体联系图如图 2-43 所示。



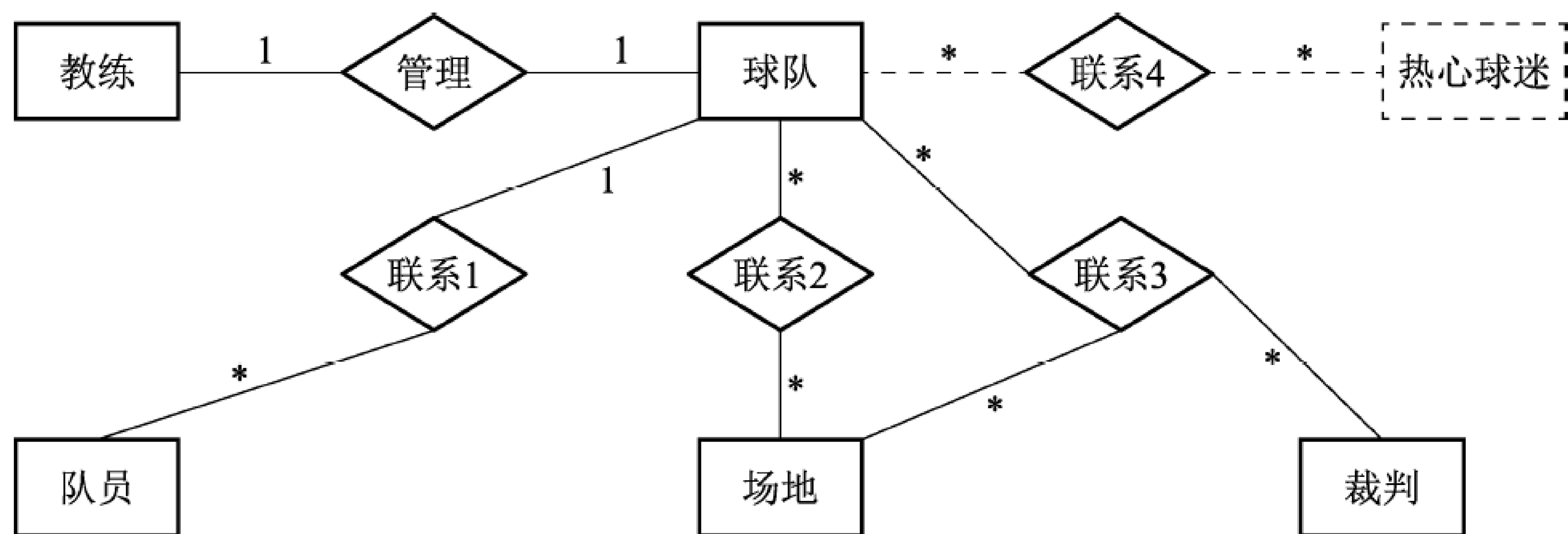


图 2-43　修改后的实体联系图

修改后的关系模式如下：

热心球迷(球迷编号，姓名，住址，俱乐部)；

支持球队(球迷编号，球队)。

11.

【问题 1】

(1) n 或 m 或*；(2) 1；(3) n 或 m 或*；(4) n 或 m 或*。

【问题 2】

完整的实体联系图如图 2-44 所示。

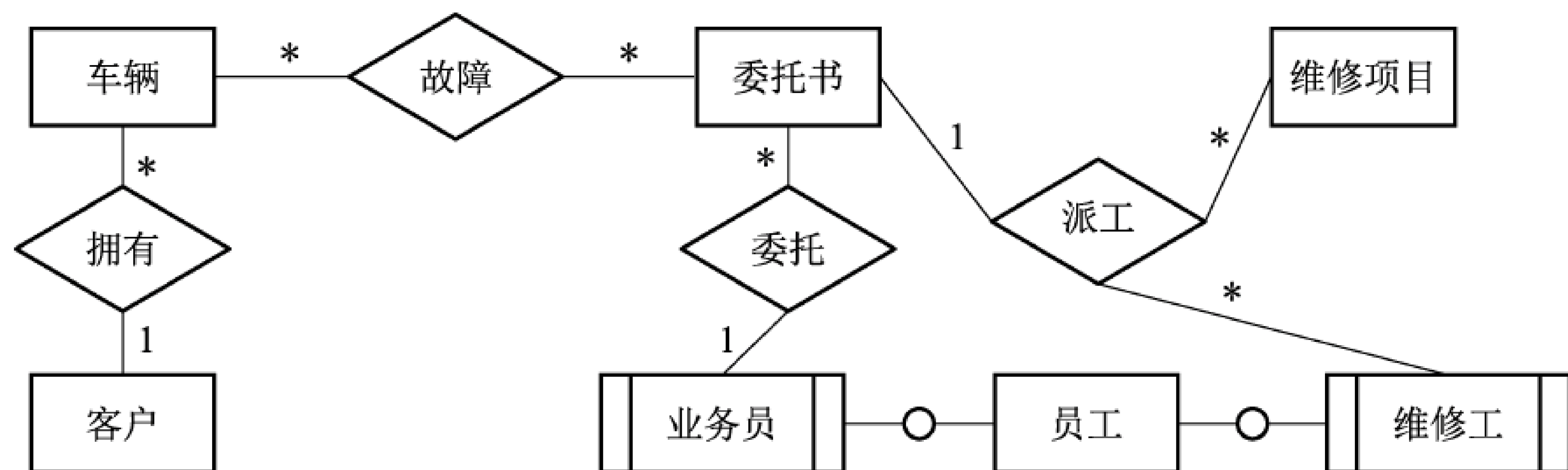


图 2-44　完整的实体联系图

【问题 3】

(5) 客户编号，客户名称，客户性质；

(6) 委托书编号，客户编号，车牌号，业务员编号；

或：委托书编号，车牌号，业务员编号。

(7) 委托书编号，维修工编号，维修项目编号；

(8) 员工编号，员工姓名。

【问题 4】

客户模式关系的主键：客户编号。

委托书模式关系的主键：委托书编号。

派工单模式关系的主键：委托书编号，维修项目编号，维修工编号。

12.

【问题 1】

(1) 1；(2) n；(3) n；(4) n。

【问题 2】
缺少的联系数：3。
挂号单：收银员。
挂号单：医师。
挂号单：门诊处方。
【问题 3】
(5) 收银员；
(6) 就诊号；
(7) 药品编码，数量，单价；
(8) 类型，库存，货架编号，单位，规格，单价。
挂号单主键：就诊号。
门诊处方主键：就诊号。
处方明细主键：就诊号、药品编码。
13.
【问题 1】
(1) 房间号，身份证号。
【问题 2】
住宿主键：房间号，身份证号，入住日期。
住宿外键：房间号，身份证号。
【问题 3】
(2) 住宿.身份证号。
(3) Having。
(4) Order by count(入住日期)　Desc 或 Order by 2 Desc。
【问题 4】
表：住宿。
属性：入住日期。
类型：聚簇索引或聚集索引或 cluster。
原因：表中记录的物理顺序与索引项的顺序一致，根据索引访问数据时，一次读取操作可以获取多条记录数据，因而可以减少查询时间。
14.
【问题 1】
(1) n；(2) m；(3) 1；(4) n 或 m；(5) m；(6) n。
注：(1)、(2)答案可互换，(5)、(6)答案可互换。
【问题 2】
(a) 读者 ID，图书 ID；(b) 读者 ID，ISBN 号。
【问题 3】
各关系模式的主键和外键如表 2-26 所示。

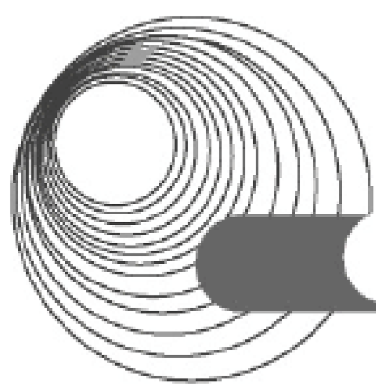


表 2-26　各关系模式的主键和外键

关系模式	主　键	外　键
读者	读者 ID	—
书目	ISBN 号	—
图书	图书 ID	ISBN 号
借还记录	读者 ID，图书 ID，借出时间	读者 ID，图书 ID
预约登记	读者 ID，ISBN 号，预约时间	读者 ID，ISBN 号，图书 ID

2.2　本 章 小 结

本章知识点在 2009 年的新大纲中改动不大。

根据近几年软件设计师水平考试试题分布情况来看，数据库设计已经成为下午部分的必考题。占的分数也是一定的，几个小问题，一共 15 分。

数据库设计的题目主要考查 E-R 模型、概念模式设计、关系模式设计、主键/外键和联系的类型等知识点，只要仔细看清题目是不难拿分的。

第 3 章　UML 分析与设计

大纲要求：

学会面向对象的分析与设计，掌握 UML 描述方法。

3.1　UML 的基础知识

3.1.1　考点辅导

3.1.1.1　面向对象的分析与设计

面向对象方法是一种运用对象、类、继承、封装、聚合、关联、消息、多态性等概念来构造系统的软件开发方法。

面向对象分析的目的是获得对应用问题的理解。理解的目的是确定系统的功能和性能要求。

面向对象分析包含 5 个活动：认定对象、组织对象、描述对象之间的相互作用、定义对象的操作、定义对象的内部信息。

面向对象设计可建立软件系统的结构。其主要工作分为两个阶段：高层设计和类设计。高层设计构造应用软件的总体模型。类设计是在标识了一个类之后给出它的规格说明，其中包括类的实例可执行的操作及其数据表示。

3.1.1.2　UML 概述

UML(Unified Modeling Language，统一建模语言)是面向对象软件的标准化建模语言，具有丰富的表达力，可以描述开发所需要的各种视图，然后以这些视图为基础装配系统。

UML 由 3 个要素构成：UML 的基本构造块、支配这些构造块如何放置在一起的规则和运用于整个语言的一些公共机制。目前，UML 已经被广泛用于企业信息系统、基于 Web 的分布式应用系统、实时嵌入式系统的建模上。

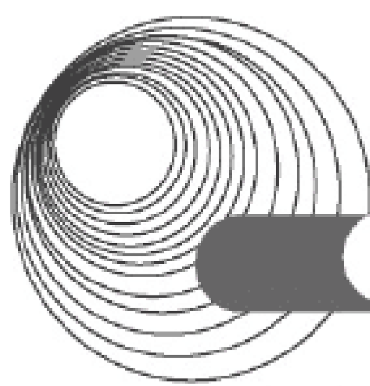
在最高层，视图被划分成 3 个视图域：结构分类、动态行为和模型管理。

(1) 结构分类描述了系统中的结构成员及其相互关系。类元包括类、用例、构件和节点。类元为研究系统动态行为奠定了基础。类元视图包括静态视图、用例视图、实现视图、部署视图。

(2) 动态行为描述了系统随时间变化的行为。行为用从静态视图中抽取的瞬间值的变化来描述。动态行为视图包括状态机视图、活动视图和交互视图。

(3) 模型管理说明了模型的分层组织结构。包是模型的基本组织单元。特殊的包还包括模型和子系统。模型管理视图跨越了其他视图并根据系统开发和配置组织这些视图。

UML 还包括多种具有扩展能力的组件，如约束、构造型和标记值，它们适用于所有的



视图元素。

UML 各种视图及其主要概念如表 3-1 所示。

表 3-1　UML 各种视图及其主要概念

主要的域	视　图	图	主要概念
结构	静态视图	类图	类、关联、泛化、依赖关系、实现、接口
	用例视图	用例图	用例、参与者、关联、扩展、包括、用例泛化
	实现视图	构件图	构件、接口、依赖关系、实现
	部署视图	部署图	节点、构件、依赖关系、实现
动态	状态机视图	状态图	状态、事件、转换、动作
	活动视图	活动图	状态、活动、完成转换、分叉、结合
	交互视图	顺序图	交互、对象、消息、激活
		协作图	协作、交互、协作角色、消息
模型管理	模型管理视图	类图	包、子系统、模型
可扩展性	所有	所有	约束、构造型、标记值

历年考题主要集中在用例图、类图、顺序图及状态图上，尤其是类图、类的属性和方法的识别，以及类间的各种关系需要重点掌握。

1. 类图

类图(Class Diagram)展现了一组对象、接口、协作及其之间的关系。在面向对象系统的建模中所建立的最常见的图就是类图。

类图给出了系统的静态设计视图，包含主动类的类图给出了系统的静态进程视图。作为模型管理视图还可以含有包或子系统，二者都用于把模型元素聚集成更大的组块。类图用于对系统的静态视图建模。这种视图主要支持系统的功能需求，即系统要提供给最终用户的服务。当对系统的静态设计建模时，通常以下述 3 种方式之一使用类图：对系统的词汇建模；对简单的协作建模；对逻辑数据库模式建模。

作为静态视图的类图可以包含依赖、关联、泛化、组合、实现关系以及注解和约束等。

(1) 依赖关系是两个事物之间的语义关系，其中一个事物发生变化会影响另一个事物的语义。

(2) 关联关系是一种结构关系，它描述了一组对象之间的链接关系。其中有一种特殊类型的关联关系，即聚集关系，它描述了整体与部分的结构关系。

(3) 泛化关系是一种一般—特殊关系，利用这种关系，子类可以共享父类的结构和行为。

(4) 实现关系是类之间的语义关系，其中的一个类制订了另一个类保证执行的契约。实现关系用于两种情况：在接口和实现它们的类或构件之间；在用例和它们的协作之间。

(5) 组合是聚集关系的变种，表示元素间更强的组合关系。各种关系图例如图 3-1 所示。

2. 用例图

用例图(Use Case Diagram)展现了一组用例、参与者(Actor)以及两者之间的关系。用例图通常包括用例、参与者、扩展关系、包含关系。用例图用于对系统的静态用例视图进行

建模，主要支持系统的行为，即该系统在它的周边环境的语境中所提供的外部可见服务。当对系统的静态用例视图建模时，可以用下列两种方式来使用用例图：对系统的语境建模；对系统的需求建模。

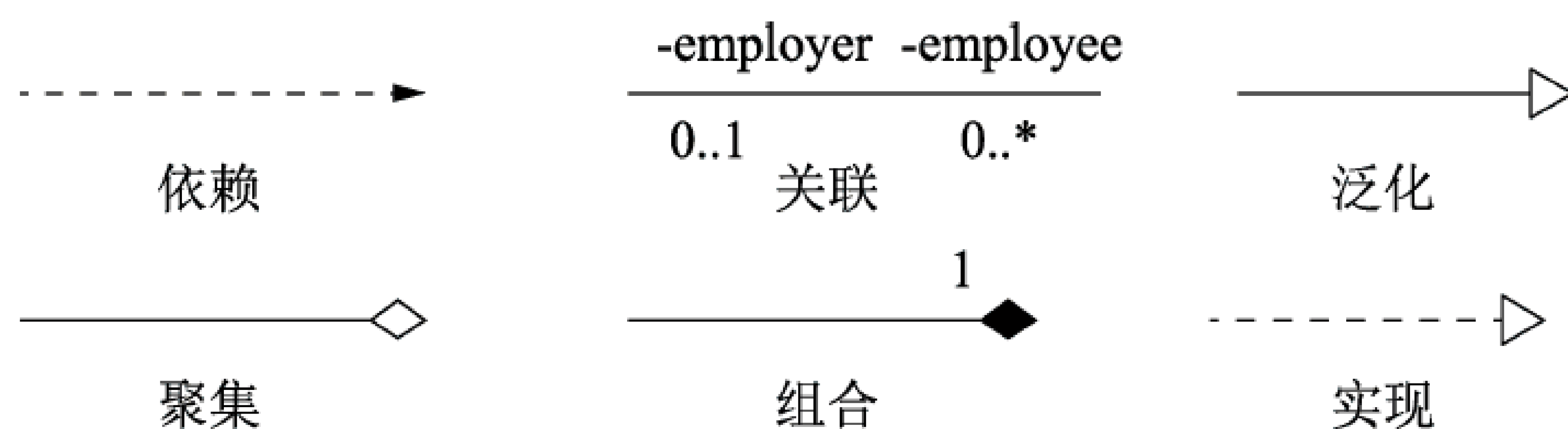


图 3-1　各种关系图例

3. 构件图

构件图(Component Diagram)展现了一组构件之间的组织和依赖关系。构件图专注于系统的静态实现视图。它与类图相关，通常把构件映射为一个或多个类、接口或协作。

4. 部署图

部署图(Deployment Diagram)展现了运行处理节点以及其中构件的配置。部署图给出了体系结构的静态实施视图。它与构件图相关，通常一个节点包含一个或多个构件。

5. 状态图

状态图(State Diagram)展现了一个状态机，它由状态、转换、事件和活动组成。

状态图关注系统的动态视图，它对接口、类和协作的行为建模尤为重要，强调对象行为的事件顺序。状态图通常包含简单状态和组合状态、转换(事件和动作)。

可以用状态图对系统的动态方面建模。这些动态方面可以包括出现在系统体系结构的任何视图中的任何一种对象的按事件排序的行为，这些对象包括类(主动类)、接口、构件和节点。

6. 活动图

活动图(Activity Diagram)是一种特殊的状态图，它展现了在系统内从一个活动到另一个活动的流程。

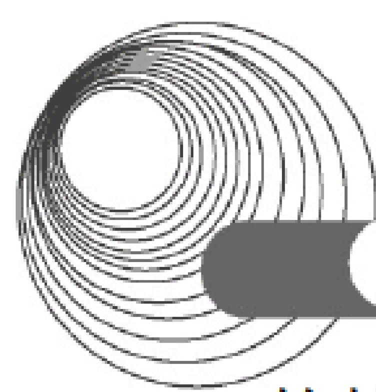
活动图专注于系统的动态视图，它对于系统的功能建模特别重要，并强调对象间的控制流程。活动图一般包括活动状态和动作状态、转换和对象。当对一个系统的动态方面进行建模时，通常有两种使用活动图的方式：对工作流建模；对操作建模。

7. 交互图

顺序图(或称序列图)和协作图均被称为交互图，用于对系统的动态方面进行建模。一张交互图显示的是一个交互，由一组对象及其之间的关系组成，包含它们之间可能传递的消息。

顺序图是强调消息时间序列的交互图，协作图则是强调接收和发送消息的对象的结构组织的交互图。

交互图用于对一个系统的动态方面建模。在大多数情况下，它包括对类、接口、构件和节点的具体的或原型化的实例及其之间传递的消息进行建模。交互图可以单独使用，用于可视化、详述、构造和文档化一个特定的对象群体的动态方面，也可以用来对一个用例



的特定控制流进行建模。

序列图有以下两个不同协作图的特征。

(1) 序列图有对象生命线，对象生命线是一条垂直的虚线，表示一个对象在一段时间内存在。

(2) 序列图有控制焦点，控制焦点是一个瘦高的矩形，表示一个对象执行一个动作所经历的时间段，既可以是直接执行，也可以是通过下级过程执行。

协作图有以下两个不同于序列图的特征。

(1) 协作图有路径。

(2) 协作图有顺序号。

序列图和协作图是同构的，它们之间可以互相转换。

3.1.2　典型例题分析

例 1　某软件公司欲设计实现一个虚拟世界仿真系统。系统中的虚拟世界用于模拟现实世界中的不同环境(由用户设置并创建)，用户通过操作仿真系统中的 1~2 个机器人来探索虚拟世界。机器人维护两个变量 b1 和 b2，用来保存从虚拟世界中读取的字符。(2016 年 5 月试题三)

该系统的主要功能描述如下。

(1) 机器人探索虚拟世界(Run Robots)。用户使用编辑器(Editor)编写文件以设置想要模拟的环境，将文件导入系统(Load File)从而在仿真系统中建立虚拟世界(Setup World)。机器人在虚拟世界中的行为也在文件中进行定义，建立机器人的探索行为程序(Setup Program)。机器人在虚拟世界中探索时(Run Program)，有两种运行模式。

① 自动控制(Run)：事先编排好机器人的动作序列(指令(Instruction))，执行指令，使机器人可以连续动作。若干条指令构成机器人的指令集(Instruction Set)。

② 单步控制(Step)：自动控制方式的一种特殊形式，只执行指定指令中的一个动作。

(2) 手动控制机器人(Manipulate Robots)。选定 1 个机器人(Select Robot)后，可以采用手动方式控制它。手动控制有 4 种方式。

① Move：机器人朝着正前方移动一个交叉点。

② Left：机器人原地沿逆时针方向旋转 90 度。

③ Read：机器人读取其所在位置的字符，并将这个字符的值赋给 b1；如果这个位置上没有字符，则不改变 b1 的当前值。

④ Write：将 b1 中的字符写入机器人当前所在的位置，如果这个位置上已经有字符，该字符的值将会被 b1 的值替代。如果这时 b1 没有值，即在执行 Write 动作之前没有执行过任何 Read 动作，那么需要提示用户相应的错误信息(Show Errors)。

手动控制与单步控制的区别在于，单步控制时执行的是指令中的动作，只有一种控制方式，即执行下个动作；而手动控制时有 4 种动作。

现采用面向对象方法设计并实现该仿真系统，得到如图 3-2 所示的用例图和图 3-3 所示的初始类图。图 3-3 中的类“Interpreter”和“Parser”用于解析描述虚拟世界的文件以及机器人行为文件中的指令集。

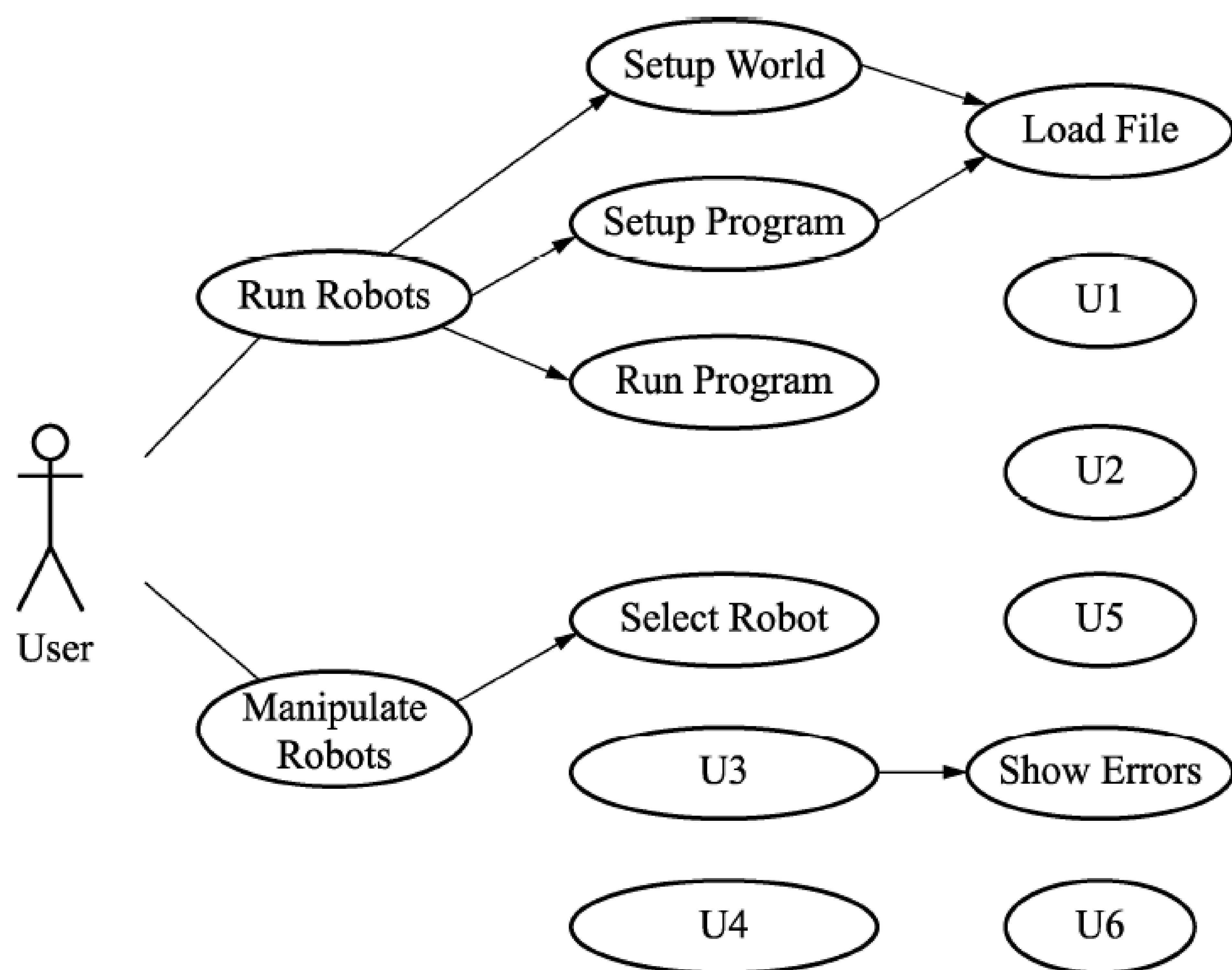


图 3-2　用例图

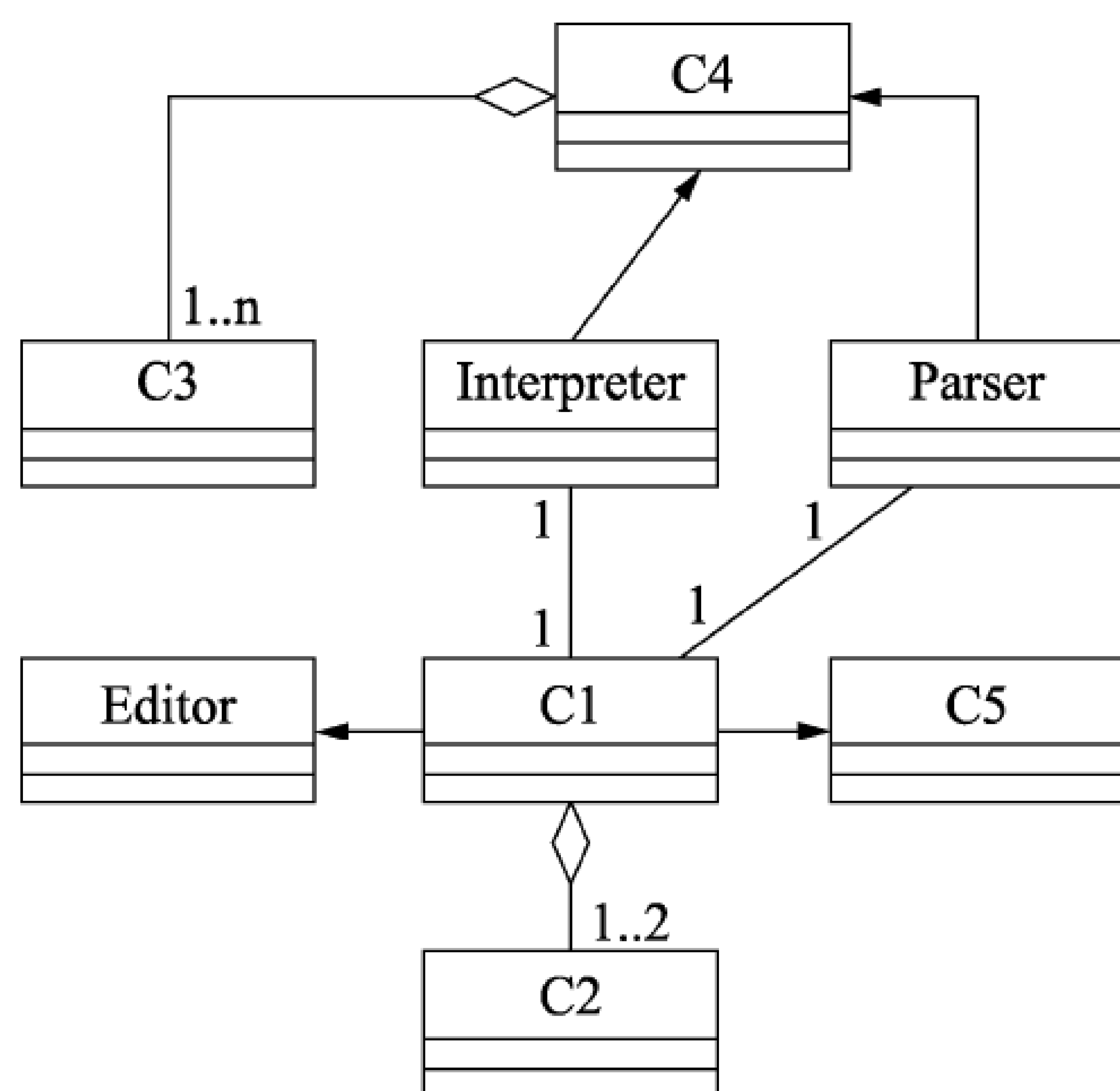


图 3-3　初始类图

【问题 1】(6 分)

根据说明中的描述，给出图 3-2 中 U1~U6 所对应的用例名。

【问题 2】(4 分)

图 3-2 中用例 U1~U6 分别与哪个(哪些)用例之间有关系，是何种关系？

【问题 3】(5 分)

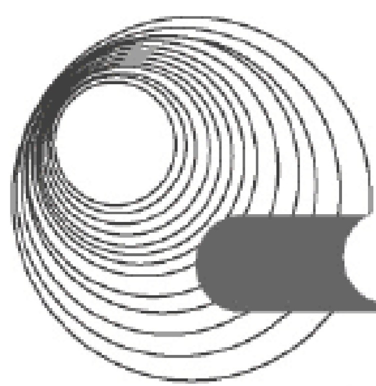
根据说明中的描述，给出图 3-3 中 C1~C5 所对应的类名。

解析：

本题考查面向对象分析中的用例图和类图，用例图描述了一组用例、参与者及它们之间的关系，包括以下几个部分：用例(Case)、参与者(Actor)。

【问题 1】

根据系统的主要功能描述可知，机器人在虚拟世界中探索时(Run Program)，有两种运



行模式：自动控制(Run)和单步控制(Step)。所以 U1/U2 为 Run、Step；根据手动控制机器人(Manipulate Robots)，手动控制有 4 种方式，由描述可知 U3 为 Write，U4/U5/U6 为 Move、Left、Read。

【问题 2】

UML 用例图中有包含(include)、扩展(extend)和泛化(generalization)三种关系。

用例 U1 和 U2 是“Run Program”用例的继承，且表现出更特别的行为，所以 U1 和 U2 与“Run Program”之间是实际应用中很少使用的泛化关系；U3~U6 用例是“Select Robot”用例中相对独立且可选的动作，是对“Select Robot”用例的功能扩展，即扩展关系。

【问题 3】

由图 3-3 中的类“Interpreter”和“Parser”是用于解析描述虚拟世界的文件以及机器人行为文件中的指令集可知，C4 为机器人的指令集(Instruction Set)，C3 为机器人动作序列(指令(Instruction))；由功能描述中用户使用编辑器(Editor)编写文件，可知 C1 为文件类，由 C2 类聚集到 C1 即上面分析的文件类，可知 C2 为机器人在虚拟世界的行为，C5 则表示整个仿真系统。

答案：

【问题 1】

U1/U2：Run、Step。

U3：Write。

U4/U5/U6：Move、Left、Read。

【问题 2】

U1、U2 和 Run Program 有泛化关系。

U3、U4、U5、U6 和 Select Robot 有扩展关系。

【问题 3】

C1：文件。

C2：机器人在虚拟世界的行为。

C3：Instruction。

C4：Instruction Set。

C5：仿真系统。

例 2 某出版社拟开发一个在线销售各种学术出版物的网上商店(ACShop)，其主要功能需求描述如下。(2015 年 11 月试题三)

(1) ACShop 在线销售的学术出版物包括论文、学术报告或讲座资料等。

(2) ACShop 的客户分为两种：未注册客户和注册客户。

(3) 未注册客户可以浏览或检索出版物，将出版物添加到购物车中。未注册客户进行注册操作之后，成为 ACShop 注册客户。

(4) 注册客户登录之后，可将待购买的出版物添加到购物车中，并进行结账操作。结账操作的具体流程描述如下。

① 从预先填写的地址列表中选择一个作为本次交易的收货地址。如果没有地址信息，则可以添加新地址。

② 选择付款方式。ACShop 支持信用卡付款和银行转账两种方式。注册客户可以从预

先填写的信用卡或银行账号中选择一个付款。若没有付款方式信息，则可以添加新付款方式。

③ 确认提交购物车中待购买的出版物后，ACShop 会自动生成与之相对应的订单。

(5) 管理员负责维护在线销售的出版物目录，包括添加新出版物或者更新在售出版物信息等操作。

现采用面向对象方法分析并设计该网上商店 ACShop，得到如图 3-4 所示的用例图和图 3-5 所示的类图。

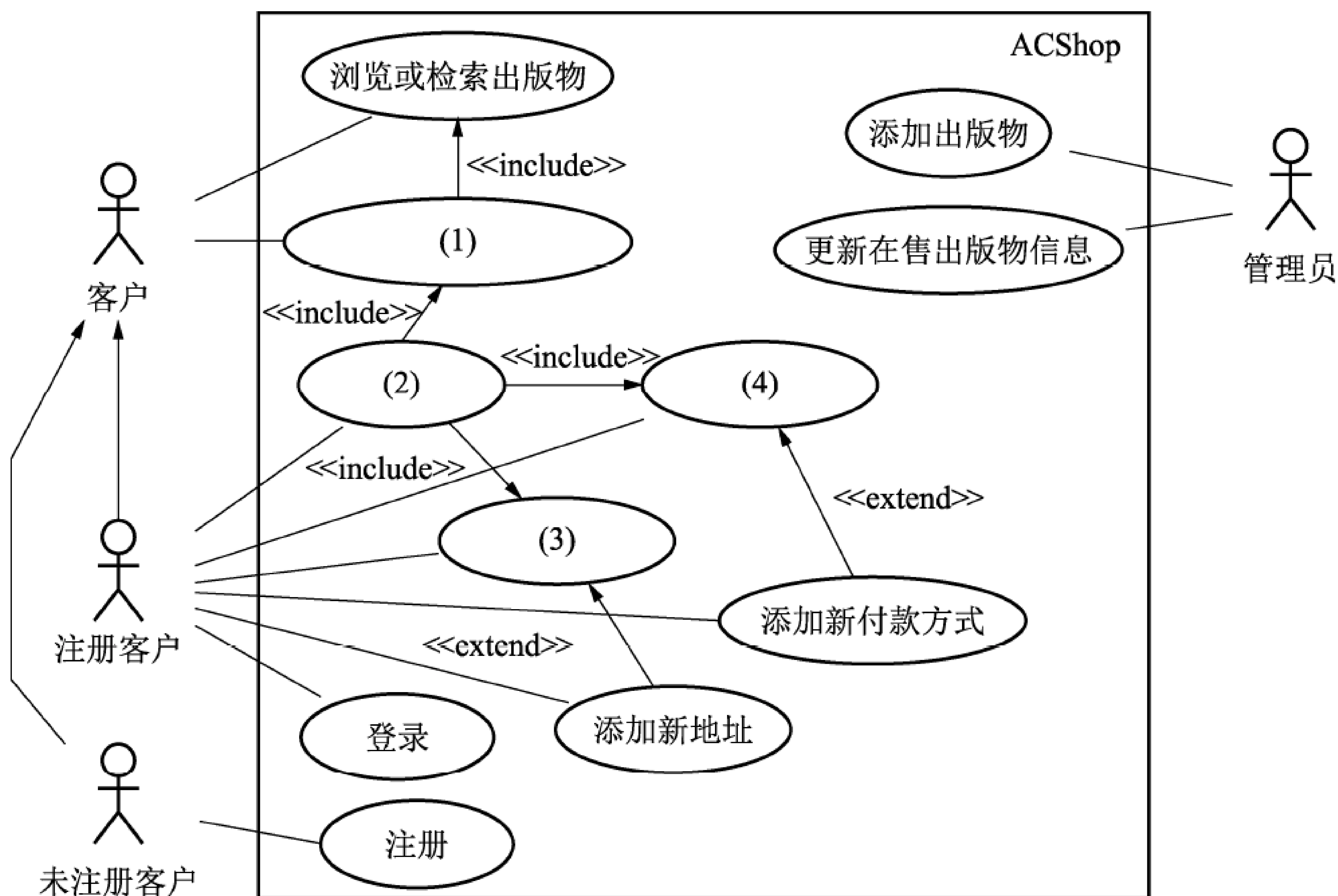


图 3-4　用例图

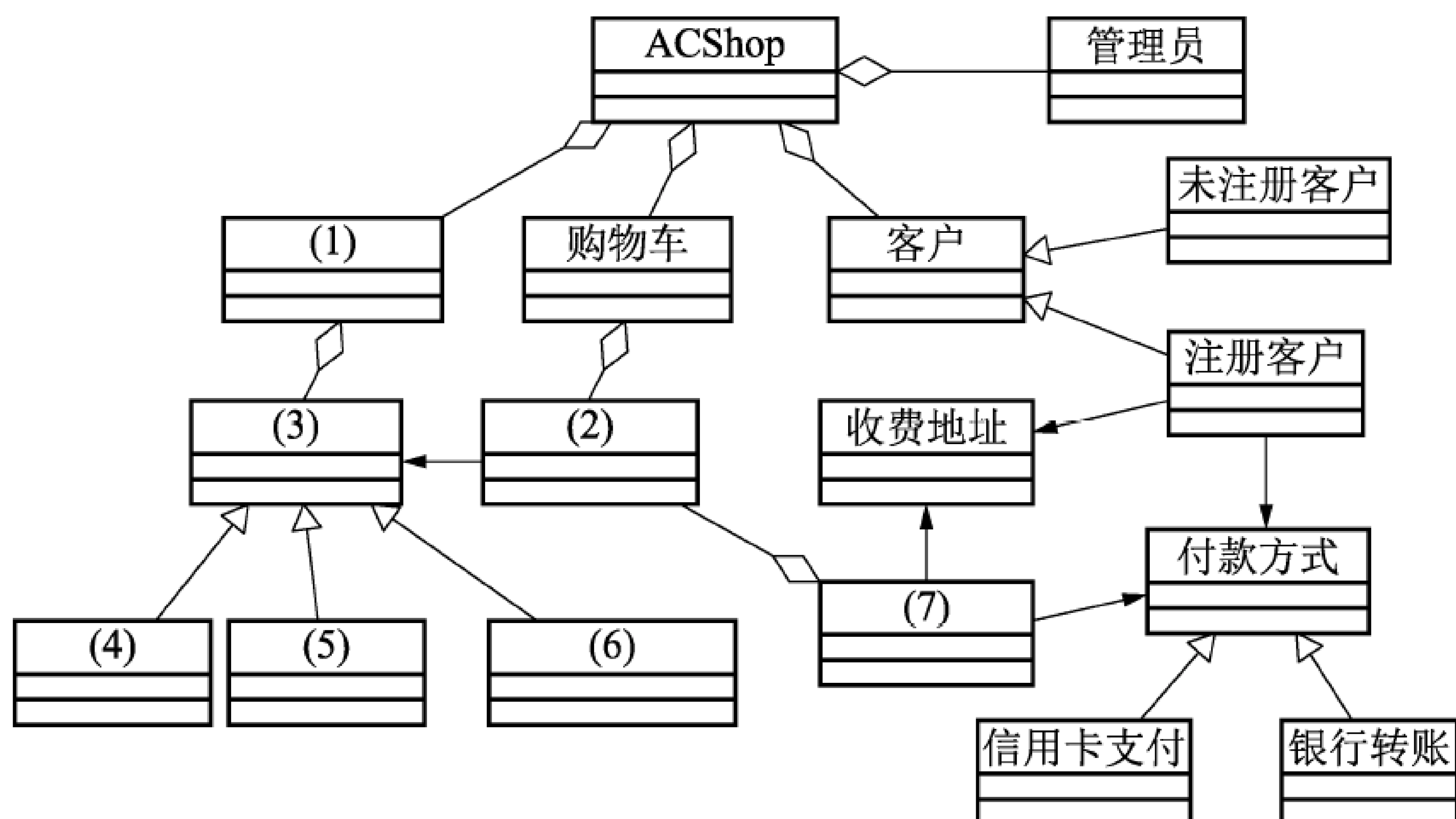
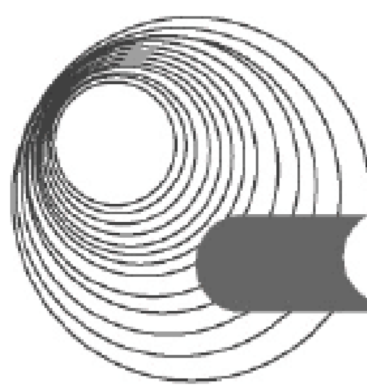


图 3-5　类图

【问题 1】(4 分)

根据说明中的描述，给出图 3-4 中(1)~(4)所对应的用例名。



【问题 2】(4 分)

根据说明中的描述，分别说明用例“添加新地址”和“添加新付款方式”会在何种情况下由图 3-4 中的用例(3)和(4)扩展而来?

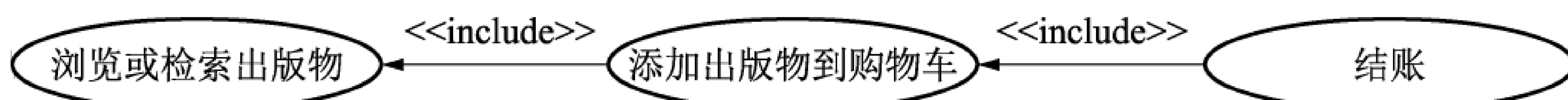
【问题 3】(7 分)

根据说明中的描述，给出图 3-5 中(1)~(7)所对应的类名。

解析：

【问题 1】

未注册客户可以浏览或检索出版物，将出版物添加到购物车中；而注册客户在将出版物添加到购物车之后还可以进行结账操作，因此有扩展关系：



而结账又分多个步骤：选择收货地址，选择付款方式。如果没有收货地址，则添加新地址；如果没有付款方式信息，可以添加新的付款方式，显然空(3)、空(4)所对应的用例为“选择收货地址”和“选择付款方式”。

【问题 2】

题干中，在描述具体结账流程时指出，在选择收货地址时，如果没有地址信息，则可以添加新地址；在选择付款方式时，若没有付款方式信息，则可以添加新付款方式。

【问题 3】

类图中，——→ 表示关联关系，这是一种拥有的关系，它使一个类知道另一个类的属性和方法，如老师与学生，丈夫与妻子。关联可以是双向的，也可以是单向的。双向的关联可以有两个箭头或者没有箭头，单向的关联有一个箭头。——▷表示泛化关系，这是一种继承关系，表示一般与特殊的关系，它指定了子类如何特化父类的所有特征和行为。——◇表示聚合，这是整体与部分的关系，且部分可以离开整体而单独存在。如车和轮胎是整体和部分的关系，轮胎离开车仍然可以存在。

网上商店由出版物目录、管理员、购物车和客户组成。放在购物车中的是“待购买的出版物”；出版物目录是由在线销售的所有学术出版物组成的，而学术出版物又分为论文、学术报告或讲座资料等，得到答案并不难。

答案：

【问题 1】

(1) 添加出版物到购物车。

(2) 结账。

(3) 选择收货地址。

(4) 选择付款方式。

【问题 2】

当选择收货地址时，若没有地址信息，则使用扩展用例“添加新地址”来完成新地址的添加。

当选择付款方式时，若没有付款方式信息，则使用扩展用例“添加新付款方式”来完成新付款方式的添加。

【问题 3】

(1) 出版物目录。

(2) 待购买的出版物。

(3) 学术出版物。

(4)~(6) 论文、学术报告、讲座资料。

(7) 订单。

例 3　某物品拍卖网站为参与者提供物品拍卖平台，组织拍卖过程，提供在线或线下交易服务。网站主要功能描述如下。(2015 年 5 月试题三)

(1) 拍卖参与者分为个人参与者和团体参与者两种。不同的团体也可以组成新的团体参与拍卖活动。网站记录每个参与者的名称。

(2) 一次拍卖中，参与者或者是买方，或者是卖方。

(3) 一次拍卖只拍出来自一个卖方的一件拍卖品；多个买方可以出价；卖方接受其中一个出价作为成交价，拍卖过程结束。

(4) 在拍卖结算阶段，买卖双方可以选择两种成交方式：线下成交，买卖双方在事先约定好的成交地点，当面完成物价款的支付和拍卖品的交付；在线成交，买方通过网上支付平台支付物价款，拍卖品由卖方通过快递邮寄给买方。

一次拍卖过程的基本事件流描述如下。

(1) 卖方在网站上发起一次拍卖，并设置本次拍卖的起拍价。

(2) 确定拍卖标的以及拍卖标的保留价(若在拍卖时间结束时，所有出价均低于拍卖标的保留价，则本次拍卖失败)。

(3) 在网站上发布本次拍卖品的介绍。

(4) 买方参与拍卖，给出竞拍价。

(5) 卖方选择接受一个竞拍价作为成交价，结束拍卖。

(6) 系统记录拍卖成交价，进入拍卖结算阶段。

(7) 卖方和买方协商拍卖品成交方式，并完成成交。

现采用面向对象方法对系统进行分析与设计，得到如表 3-2 所示的类列表以及如图 3-6 所示的类图，类中关键属性与方法如表 3-3 所示。

表 3-2　物品拍卖网站类列表

序　号	类　名	说　明
C1	SellerRole	一次拍卖中的卖方
C2	Item	拍卖品
C3	Auction	拍卖活动
C4	Sale	拍卖结算
C5	AuctionParticipant	拍卖参与者
C6	Interchange	成交方式
C7	OneParticipant	个人参与者
C8	OfflinePay	线下成交
C9	CompositeParticipant	团体参与者

续表

序　号	类　名	说　明
C10	OnlinePay	在线成交
C11	Bid	拍卖标的
C12	BuyerRole	一次拍卖中的买方

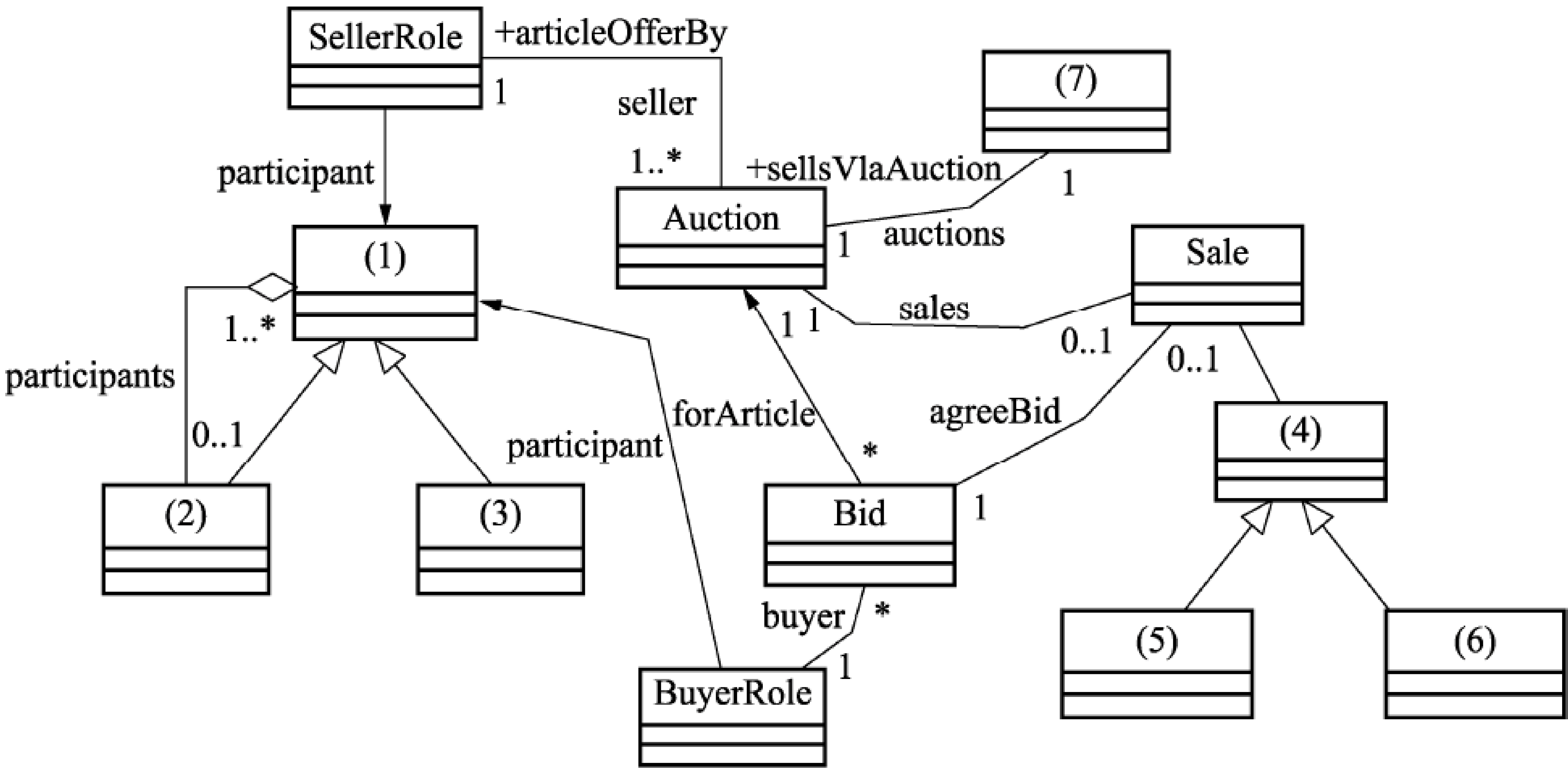


图 3-6　类图

表 3-3　关键属性与方法列表

序　号	名　称	说　明
M1	name	属性名，用户名称
M2	description	属性名，拍卖品描述
M3	minBidPrice	属性名，拍卖的起拍价
M4	agreePrice	属性名，拍卖成交价
M5	bidPrice	属性名，拍卖标的保留价
M6	address	属性名，线下成交地点
M7	sellerAccount	属性名，卖方网上支付账户名
M8	buyerAddress	属性名，买方邮寄地址
M9	placeBidForAuction	方法名，为拍卖品出竞拍价
M10	sellNewArticle	方法名，发起一次拍卖

【问题 1】(7 分)

根据说明中的描述，给出图 3-6 中(1)~(7)所对应的类名(类名使用表 3-2 中给出的序号)。

【问题 2】(5 分)

根据说明中的描述，确定表 3-3 中的属性/方法分别属于哪个类(类名、方法/属性名使用表 3-2、表 3-3 中给出的序号)。

【问题 3】(3 分)

图 3-6 采用了何种设计模式？以 100 字以内文字说明采用这种设计模式的原因。

解析：

【问题 1】

参与拍卖的为“参与拍卖者”，拍卖参与者又可分为个人参与者和团体参与者两种，显然(1)为“参与拍卖者”，属于类 C5，(2)为“团体参与者”，属于类 C9，(3)为“个人参与者”，属于类 C7。拍卖的对象为拍卖品，(7)为“拍卖品”，属于类 C2。拍卖成交后，在结算时有两种成交方式：线下成交和在线成交，显然(4)为“成交方式”，属于类 C6，(5)、(6)为“线下成交”，属于类 C8、“在线成交”，属于类 C10。

【问题 2】

“用户名称”name 属性应属于“拍卖参与者”类 C5；“拍卖品描述”description 属性应属于“拍卖品”类 C2；“拍卖的起拍价”minBidPrice 属性、“拍卖成交价”agreePrice 属性、“拍卖标的保留价”bidPrice 属性应属于“拍卖活动”类 C3；“线下成交地点”address 属性应属于“线下成交”类 C8；“卖方网上支付账户名”sellerAccount 属性、“买方邮寄地址”buyerAddress 属性应属于“在线成交”类 C10；“为拍卖品出竞拍价”placeBidForAuction 方法应属于“一次拍卖中的买方”类 C12；“发起一次拍卖”sellNewArticle 方法应属于“一次拍卖中的卖方”类 C1。

【问题 3】略。

答案：

【问题 1】

(1)C5。(2)C9。(3)C7。(4)C6。(5)(6)C8、C10。(7)C2。

【问题 2】

M1：C5。M2：C2。M3：C3。M4：C3。M5：C3。M6：C8。M7：C10。M8：C10。M9：C12。M10：C1。

【问题 3】

组合模式，在本题中由于拍卖者分为个人参与者和团体参与者两种，而团体也可以组成新的团体参与拍卖活动。这样的整体—部分关系，适合于使用组合模式表达。

例 4　某公司欲开发一个管理选民信息的软件系统。系统的基本需求描述如下。(2014 年 11 月试题三)

(1) 每个人(Person)可以是一个合法选民(Eligible)或者无效选民(Ineligible)。

(2) 每个合法选民必须通过该系统对其投票所在区域(即选区，Riding)进行注册(Registration)。每个合法选民仅能注册一个选区。

(3) 选民所属选区由其居住地址(Address)决定。假设每个人只有一个地址，地址可以是镇(Town)或者城市(City)。

(4) 某些选区可能包含多个镇；而某些较大的城市也可能包含多个选区。

现采用面向对象方法对该系统进行分析与设计，得到如图 3-7 所示的初始类图。

【问题 1】(8 分)

根据说明中的描述，给出图 3-7 中 C1~C4 所对应的类名(类名使用说明中给出的英文词汇)。

【问题 2】(3 分)

根据说明中的描述，给出图 3-7 中 M1~M6 处的多重度。

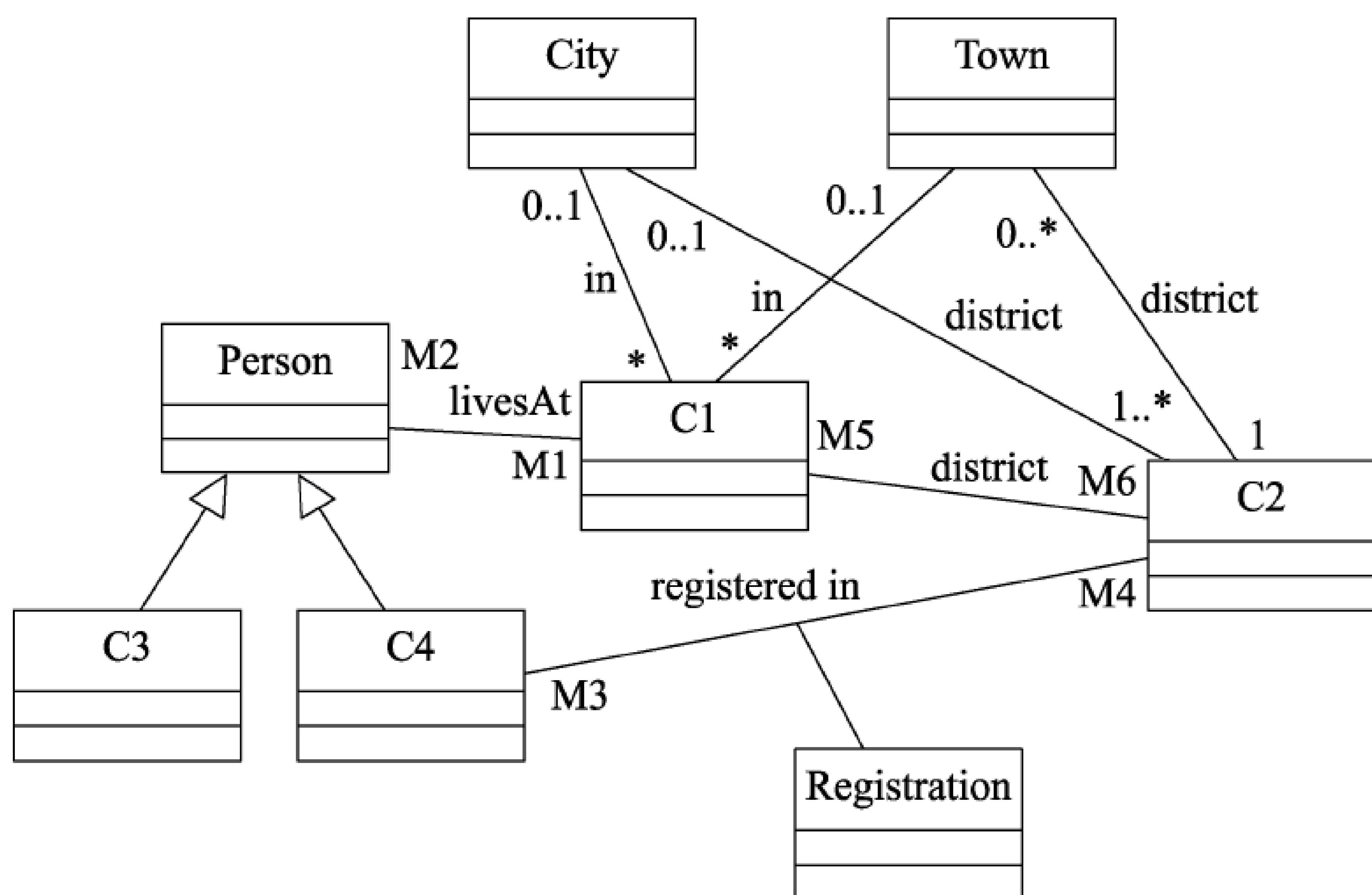


图 3-7　类图

【问题 3】(4 分)

现对该系统提出了以下新需求。

(1) 某些人拥有在多个选区投票的权利，因此需要注册多个选区。

(2) 对于满足(1)的选民，需要划定其“主要居住地”，以确定他们应该在哪个选区进行投票。

为了满足上述需求，需要对图 3-7 所示的类图进行哪些修改？请用 100 字以内文字说明。

解析：

【问题 1】

由“每个人可以是一个合法选民或者无效选民”可知 C3 和 C4 是这两者中的一个，由 C4 和 C2 关联可知，C4 为合法选民(Eligible)，则 C3 为无效选民(Ineligible)。由 City 和 Town 共同指向 C1 以及描述“选民所属选区由其居住地址(Address)决定。假设每个人只有一个地址，地址可以是镇(Town)或者城市(City)”可知，C1 应该为 Address。由描述“每位合法选民必须通过该系统对其投票所在区域(即选区，Riding)进行注册(Registration)”以及 C4 指向 C2 和 City、Town 同时指向 C2 可知，C2 为 Riding，即选区。

【问题 2】

UML 中关联的多重度是指一个类的实例能够与另一个类的多少个实例相关联。具体的取值意义如下。

0..1：0 个或 1 个。

1：只能 1 个。

0..*：0 个或多个。

*：0 个或多个。

1..*：1 个或多个。

由描述“每个人只有一个地址”可知，M1 为 1；一个地址可以有 0 个或多个人，因此 M2 为*；一个选区可以有 0 个或多个选民，一个选民只在一个选区投票，因此，M3 为*，M4 为 1；由“某些选区可能包含多个镇”可知每个选区可包含 0 个或多个地址，M5 应为*；

由于每个选民只有一个地址，每个地址属于一个选区，因此 M6 为 1。

【问题 3】

若对系统提出新的要求：①某些人拥有在多个选区投票的权利，因此需要注册多个选区；②对于满足(1)的选民，需要划定其“主要居住地”，以确定他们应该在哪个选区进行投票。则将 M1 与 M4 由 1 修改为 1..*。

答案：

【问题 1】

C1：Address。C2：Riding。C3：Ineligible。C4：Eligible。

【问题 2】

M1：1。M2：*。M3：*。M4：1。M5：*。M6：1。

【问题 3】

将 M1 与 M4 由 1 修改为 1..*。

例 5　某高校图书馆欲建设一个图书馆管理系统，目前已经完成了需求分析阶段的工作，功能需求均使用用例进行描述，其中用例“借书(Check Out books)”的详细描述如下。(2014 年 5 月试题三)

参与者：读者<Patron>

典型事件流：

1. 输入读者 ID。
2. 确认该读者能够借阅图书，并记录读者 ID。
3. 输入所要借阅的图书 ID。
4. 根据图书目录中的图书 ID 确认该书可以借阅，计算归还时间，生成借阅记录。
5. 通知读者图书归还时间。

重复步骤 3~5，直到读者结束借阅图书。

备选事件流：

2a. 若读者不能借阅图书，说明读者违反了图书馆的借书制度(例如，没有支付借书费用等)：

① 告知读者不能借阅，并说明拒绝借阅的原因。

② 本用例结束。

4a. 读者要借阅的书无法外借：

告知读者本书无法借阅。

回到步骤 3。

说明：图书的归还时间与读者身份有关。如果读者是教师，图书可以借阅一年；如果是学生，则只能借阅 3 个月，读者 ID 中包含读者身份信息。

现采用面向对象方法开发该系统，得到如图 3-8 所示的系统类模型(部分)，以及如图 3-9 所示的系统操作 checkOut 的通信图。

【问题 1】(8 分)

根据说明中的描述，以及图 3-8 和图 3-9，给出图 3-8 中 C1~C4 处所对应的类名(类名使用图 3-8 和图 3-9 中给出的英文词汇)。

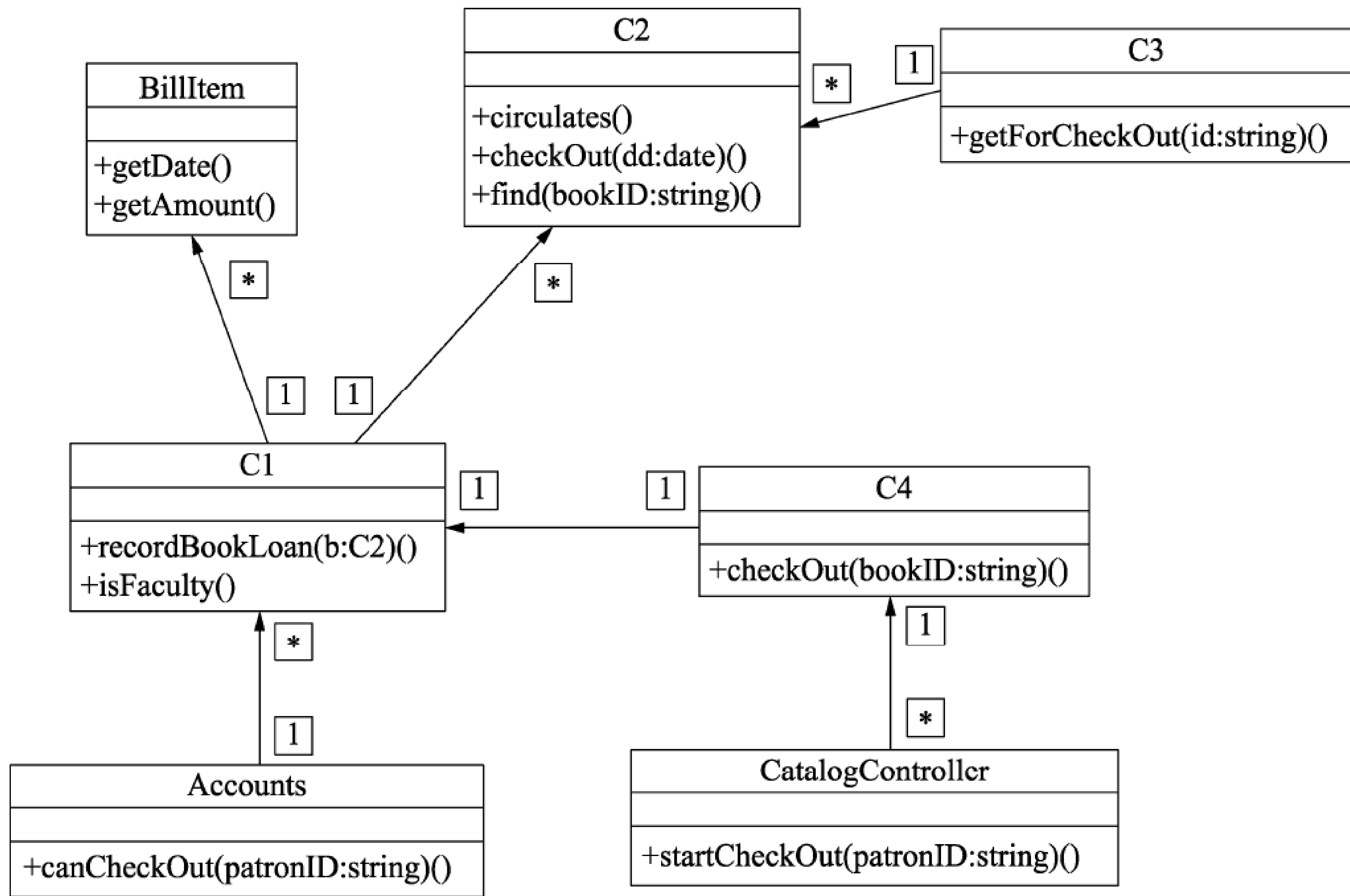


图 3-8　系统类模型

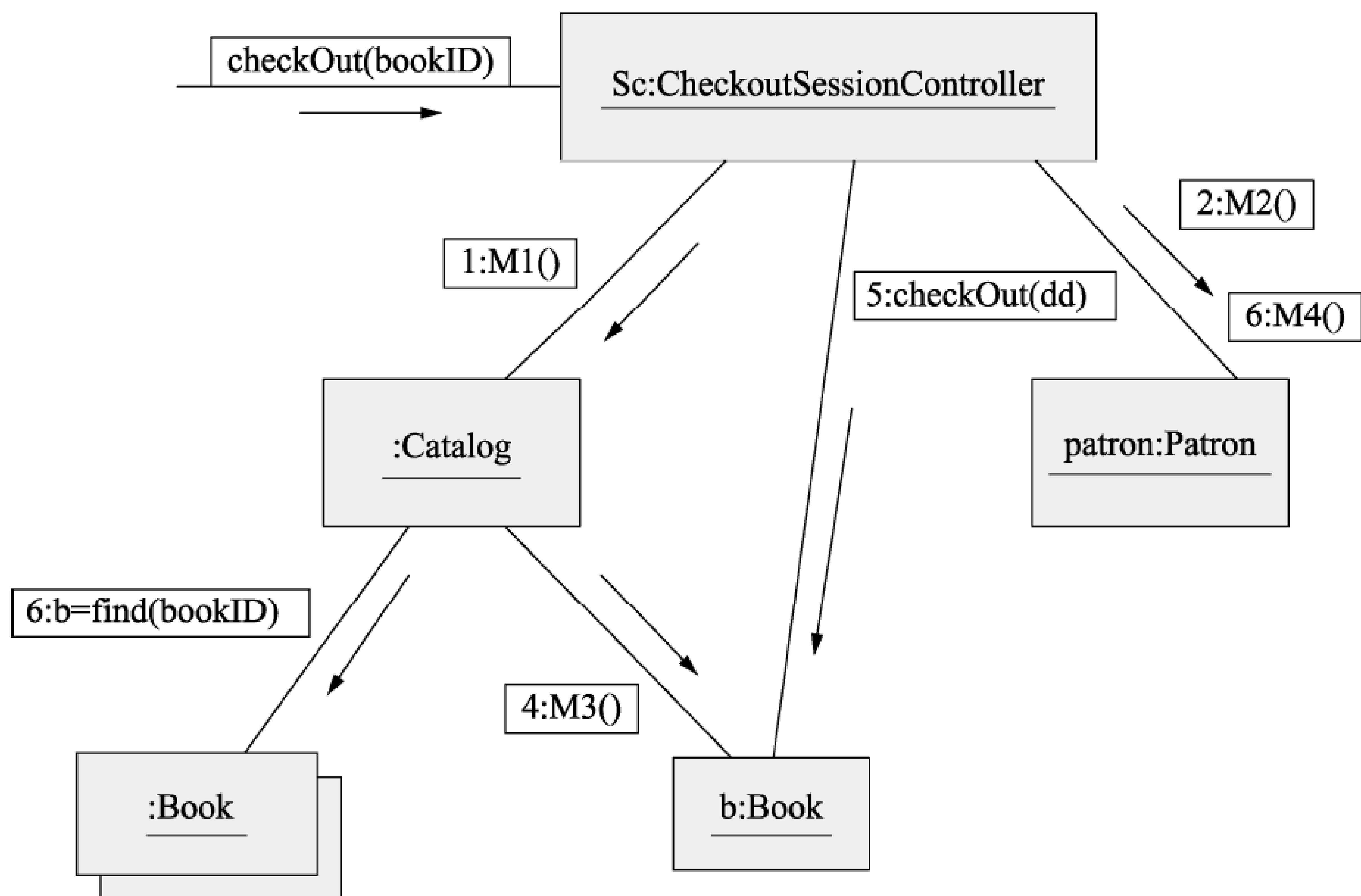


图 3-9　系统操作 checkOut 的通信图

【问题 2】(4 分)

根据说明中的描述，以及图 3-8 和图 3-9，给出图 3-9 中 M1~M4 处所对应的方法名(方法名使用图 3-8 和图 3-9 中给出的英文词汇)。

【问题 3】(3 分)

用例“借书”的备选事件流 4a 中，根据借书制度来判定读者能否借阅图书，且图书馆

的借书制度会不断地扩充，并需要根据图书馆的实际运行情况来调整具体使用哪些制度，为满足这一要求，在原有类设计的基础上，可以采用何种设计模式？简要说明原因。

解析：

【问题 1】

根据系统类模型，我们可以得出各个类之间的关联关系。

首先从类 Accounts 中的 canCheckOut(patronID:string)方法，可以看出 Accounts 关联 Patron，因此图中 C1 为 Patron。C1 为 Patron，则 C1 必会与书关联，从 C1 中的 recordBookLoad(b，C2)，可以看出 C1 关联 C2。因此 C2 为 Book。根据系统操作 checkOut 的通信图，可以看出与 Book 关联的是 Catalog，因此 C3 为 Catalog。结合两图，则可以得出 C4 为 CheckoutSessionController。

【问题 2】

结合典型事件流:

1. 输入读者 ID。

2. 确认该读者能够借阅图书，并记录读者 ID。

以上两步实际上就是判断读者是不是老师，也就是 isFaculty( )，因此 M2 为 isFaculty( )。

3. 输入所要借阅的图书 ID；对应的操作就是 M1：getForCheckOut(bookID)。

4. 根据图书目录中的图书 ID 确认该书可以借阅，计算归还时间，生成借阅记录。对应的操作就是 M3：circulates( )。

5. 通知读者图书归还时间。对应的操作就是 M4：recordBookLoan( )。

【问题 3】略。

答案：

【问题 1】

C1：Patron。C2：Book。C3：Catalog。C4：CheckoutSessionController。

【问题 2】

M1()：getForCheckOut(bookID)。M2()：isFaculty()。M3()：circulateso。M4()：recordBookLoan。

【问题 3】

应采用策略模式，策略模式的优势在于可以灵活地添加对同一问题的不同处理方案，这与题目要求非常吻合。

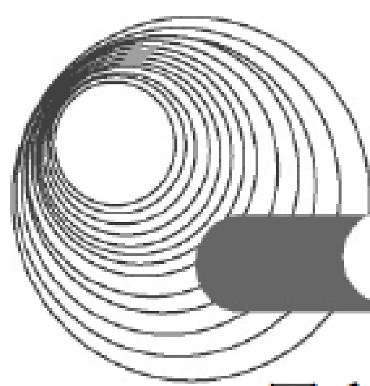
例 6　某航空公司会员积分系统(CFrequentFlyer)的主要功能描述如下。(2013 年 11 月试题三)

(1) 乘客只要办理该航空公司的会员卡，即可成为普卡会员(CBasic)。随着飞行里程数的积累，可以从普卡会员升级到银卡会员(CSilver)或金卡会员(CGold)。非会员(CNonMember)不能累积里程数。

(2) 每年年末，系统根据会员在本年度累积的里程数对下一年会员等级进行调整。

(3) 普卡会员在一年内累积的里程数若满 25 000 英里但不足 50 000 英里，则自动升级为银卡会员；若累积的里程数在 50 000 英里以上，则自动升级为金卡会员。银卡会员在一年内累积的里程数若在 50 000 英里以上，则自动升级为金卡会员。

(4) 若一年内没有达到对应级别要求的里程数，则自动降低会员等级。金卡会员一年内



累积的里程数若不足 25 000 英里，则自动降级为普卡会员；若累积的里程数达到 25 000 英里，但是不足 50 000 英里，则自动降级为银卡会员。银卡会员一年内累积的里程数若不足 25 000 英里，则自动降级为普卡会员。

采用面向对象方法对会员积分系统进行分析与设计，得到如图 3-10 所示的状态图和图 3-11 所示的类图。

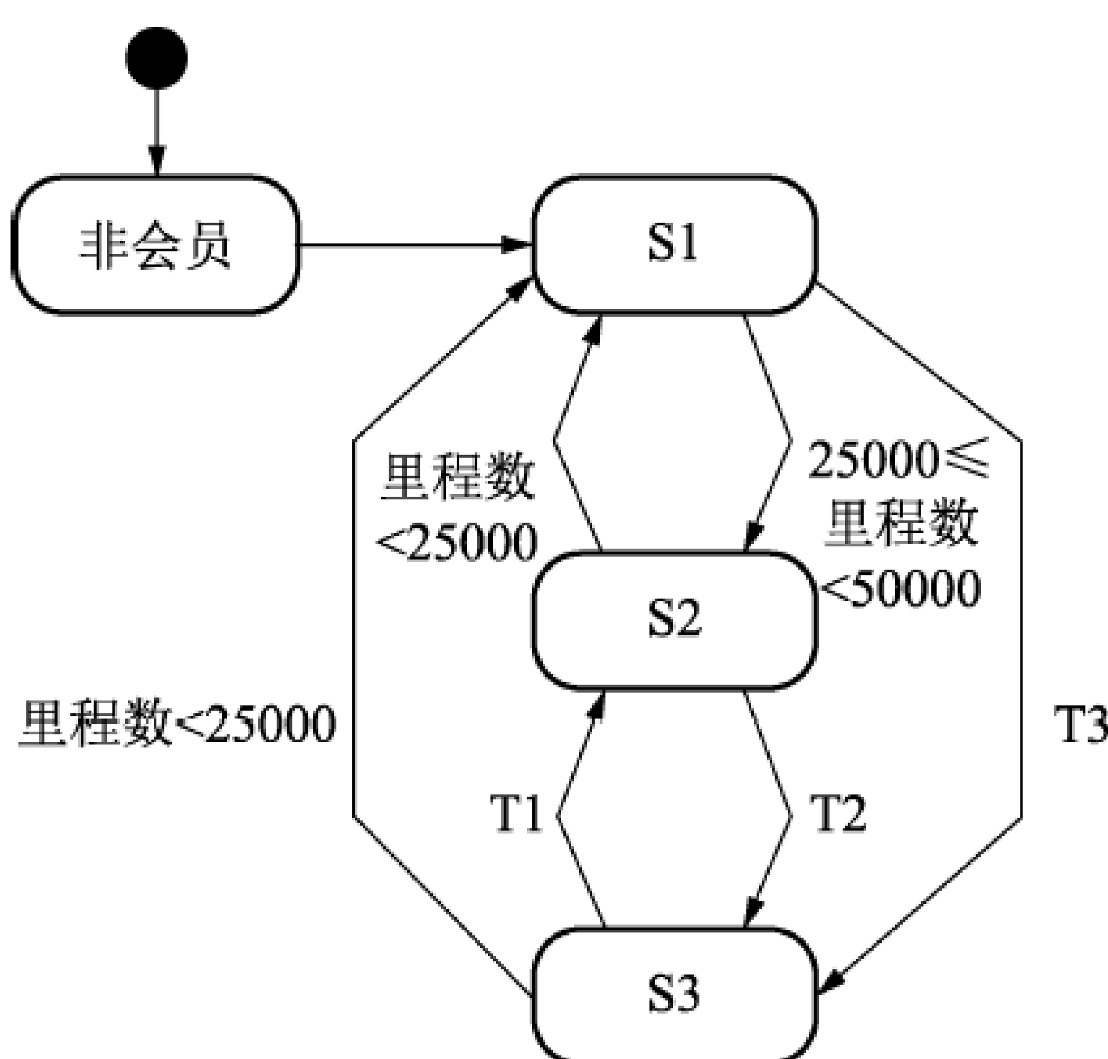


图 3-10　某会员积分系统状态图

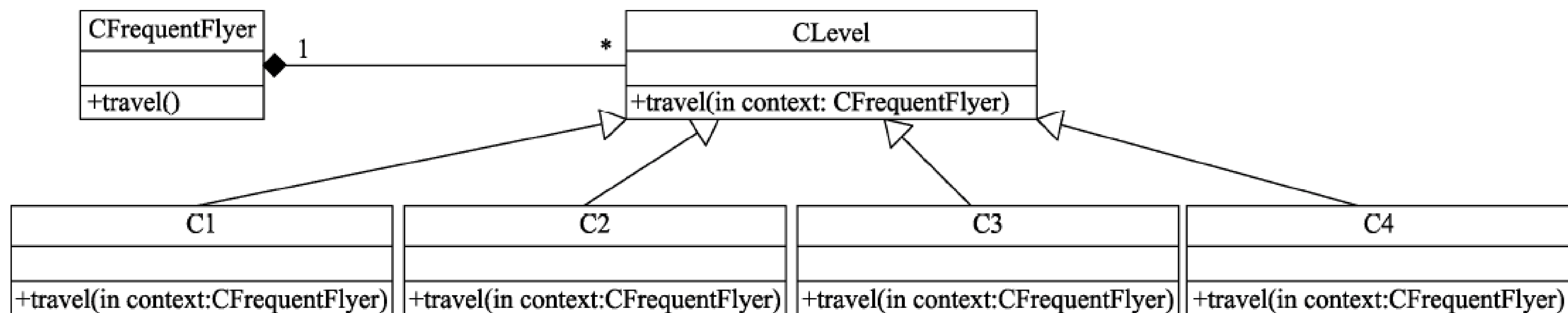


图 3-11　某会员积分系统类图

【问题 1】(6 分)

根据说明中的描述，给出图 3-10 中 S1~S3 处所对应的状态以及 T1~T3 处所对应的迁移名称。

【问题 2】(4 分)

根据说明中的描述，给出图 3-11 中 C1~C4 所对应的类名(类名使用说明中给出的英文词汇)。

【问题 3】(5 分)

图 3-11 所示的类图中使用了哪种设计模式？在这种设计模式下，类 CFrequentFlyer 必须具有的属性是什么？C1~C4 中的 travel 方法应具有什么功能?

本题考查面向对象分析中的类图、用例图。用例图描述了一组用例、参与者及它们之间的关系。包括以下几个部分：用例(Case)、参与者(Actor)。会员积分系统状态图就是一种用例图。用例视图中的参与者与系统外部的一个实体以某种方式参与了用例的执行过程；用例是一个叙述型文档，用来描述参与使用系统、完成某个事情时发生的顺序。

解析：

【问题 1】

图中要求填充 S1、S2、S3 这三个状态以及它们之间的变迁关系。本题中会员有三种状

态：普卡、金卡和银卡。普卡会员在一年内累积的里程数若满 25 000 英里但不足 50 000 英里，则自动升级为银卡会员；若累积的里程数在 50 000 英里以上，则自动升级为金卡会员。银卡会员在一年内累积的里程数若在 50 000 英里以上，则自动升级为金卡会员；所以，S1 为普卡会员，S2 为银卡会员，S3 为金卡会员。同样，根据上述分析可知，T1、T2 就是 S2 和 S3 之间的转换原则，T3 是 Sl->S2 的转换原则。

【问题 2】

本问题考查类图的层次结构和多重度。图中有 4 个非常明显的继承结构，即 C1~C4 继承 CLevel，根据说明可知代表了四类不同的会员模式。

【问题 3】

状态模式允许对象在内部状态变化时，变更其行为，并且修改其类。状态模式的类图如图 3-12 所示。

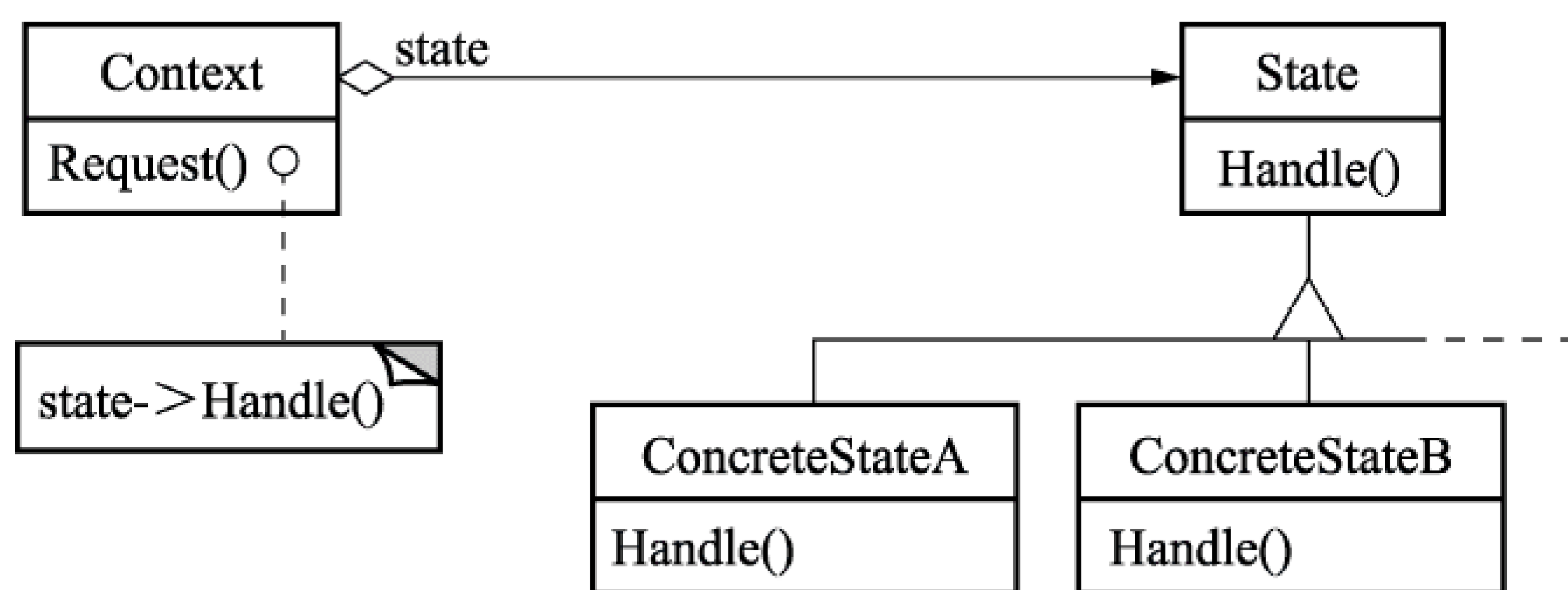


图 3-12　状态模式的类图

状态模式的组成如下。

(1) 环境类(Context)：定义客户感兴趣的接口。维护一个 ConcreteState 子类的实例，这个实例定义当前状态。

(2) 抽象状态类(State)：定义一个接口以封装与 Context 的一个特定状态相关的行为。

(3) 具体状态类(ConcreteState)：每一子类实现一个与 Context 的一个状态相关的行为。

(4) 系统类图中的类 CFrequentFlyer 对应图 3-12 中的环境类，因此类 CFrequentFlyer 应该有一个 CLevel 类的对象。

(5) Travel 方法的功能：计算飞行里程数，根据里程数判断是否需要调整会员级别(跳转到不同的状态)。

答案：

【问题 1】

S1：普卡会员。S2：银卡会员。S3：金卡会员。

T1：25 000 ≤ 里程数 < 50 000。T2：里程数 ≥ 50 000。T3：里程数 ≥ 50 000。

【问题 2】

C1：CNonMember。C2：CBasic。C3：CSilver。C4: CGold。

【问题 3】

使用了 State 模式(状态模式)。

类 CFrequentFlyer 必须具有的属性：CLevel 的对象。

Travel 方法的功能：计算飞行里程数，根据里程数判断是否需要调整会员级别(跳转到不同的状态)。

例 7 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸对应栏内。(2013 年 5 月试题三)

【说明】

某城市拟开发一个基于 Web 的城市黄页，公开发布该城市重要的组织或机构(以下统称客户)的基本信息，方便城市生活。该系统的主要功能描述如下。

(1) 搜索信息：任何使用 Internet 的网络用户都可以搜索发布在城市黄页中的信息，如客户的名称、地址、联系电话等。

(2) 认证：客户若想在城市黄页上发布信息，需通过系统的认证。认证成功后，该客户成为系统授权用户。

(3) 更新信息：授权用户登录系统后，可以更改自己在城市黄页中的相关信息，如变更联系电话等。

(4) 删除客户：对于拒绝继续在城市黄页上发布信息的客户，由系统管理员删除该客户的相关信息。

系统采用面向对象的方法进行开发，在开发过程中认定出如表 3-4 所示的类。系统的用例图和类图分别如图 3-13 和图 3-14 所示。

表 3-4 类列表

类 名	说 明
InternetClient	网络用户
CustomerList	客户集，维护城市黄页上的所有客户信息
Customer	客户信息，记录单个客户的信息
RegisteredClient	授权用户
Administrator	系统管理员

【问题 1】(5 分)

根据说明中的描述，给出图 3-13 中 A1 和 A2 处所对应的参与者、UC1 和 UC2 处所对应的用例，以及(1)处的关系。

【问题 2】(7 分)

根据说明中的描述，给出图 3-14 中 C1~C5 处所对应的类名(表 3-4 中给出的类名)和(2)~(5)处所对应的多重度。

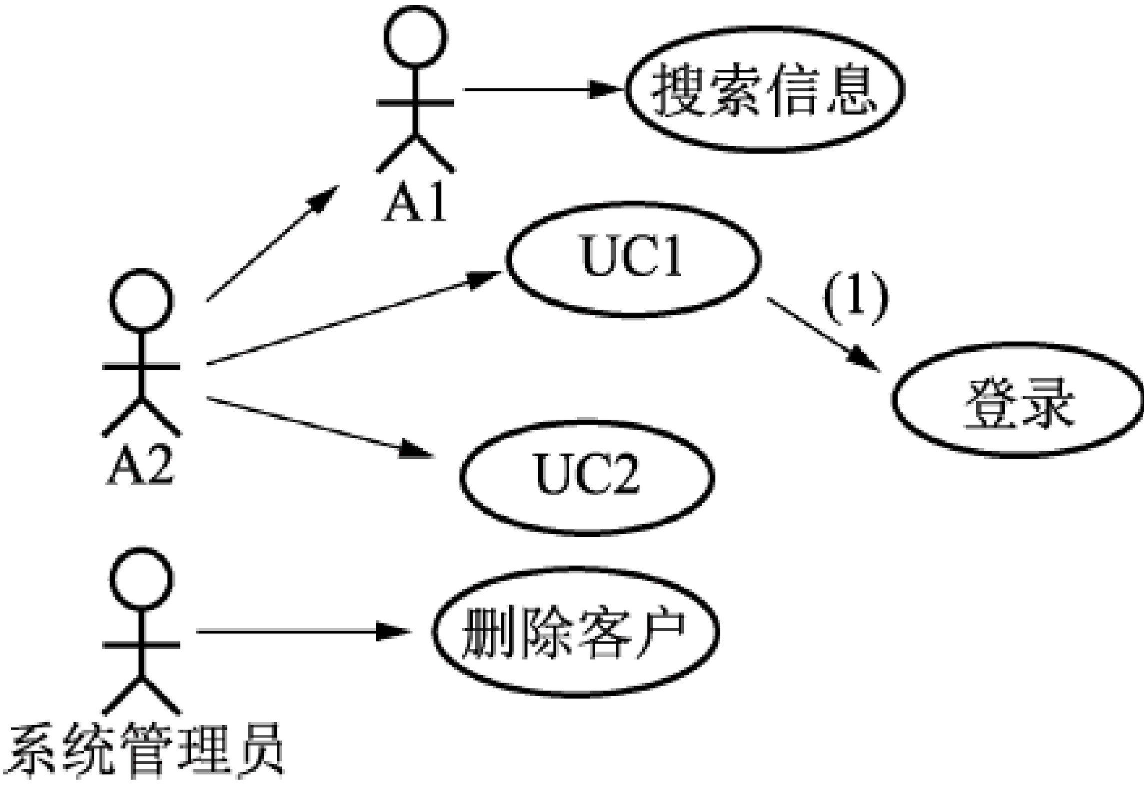


图 3-13 系统用例图

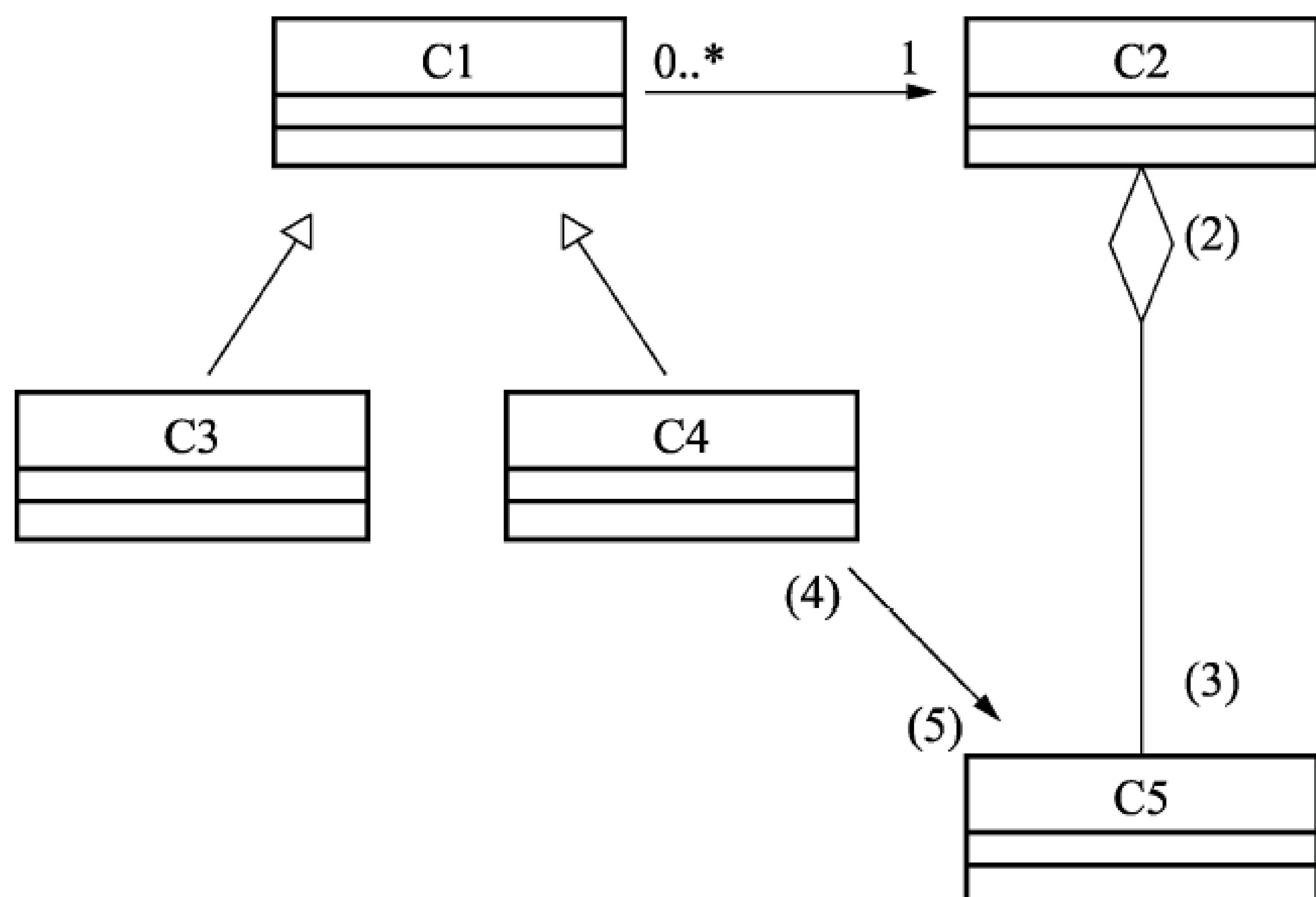


图 3-14　系统类图

【问题 3】(3 分)

认定类是面向对象分析中非常关键的一个步骤。一般首先从问题域中得到候选类集合，再根据相应的原则从该集合中删除不作为类的，剩余的就是从问题域中认定出来的类。简要说明选择候选类的原则，以及对候选类集合进行删除的原则。

解析：

本题考查面向对象分析中的类图和用例图。用例图描述了一组用例、参与者及它们之间的关系，包括以下几个部分：用例(Case)、参与者(Actor)。用例视图中的参与者与系统外部的一个实体以某种方式参与了用例的执行过程；用例是一个叙述型文档，用来描述参与使用系统、完成某个事情时发生的顺序。

【问题 1】

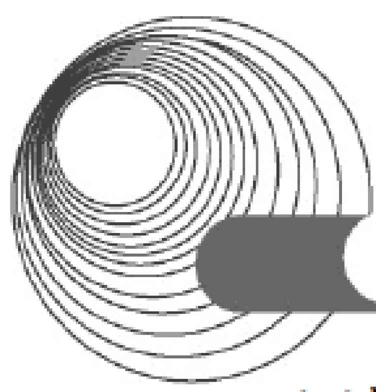
图 3-13 中，A1 可以搜索信息，A2 由 A1 派生且 A2 参与了两个用例，根据题中的说明(1)和(2)，可知 A1 为网络用户，A2 为授权用户。由用例 UC1 和“登录”用例之间存在关系，可知 UC1 为“认证”用例，因为用户登录必须先认证，所以“登录”用例是“认证”用例的扩展，它们之间的关系为 extend。对于授权用户还可以更新信息，故 UC2 为“更新信息”用例。

【问题 2】

本问题考查类图的层次结构和多重度。图 3-14 中有两个非常明显的继承结构，即 C3 和 C4 继承于 C1，且 C1 与 C2 是多对一的关系，根据说明(1)中任何网络用户都可以搜索客户信息，即 C1 为网络用户，C2 为客户信息，由此很明显地得出 C3 和 C4 只能在授权用户和系统管理员中选取。根据 C2 和 C5 之间存在聚集关系，且 C2 为客户信息，可以推断 C5 为客户集。再由 C4 和 C5 之间的关联关系，且 C5 为客户集，能对客户集进行批量操作的用户 C4 显然就是系统管理员，由此得出 C3 为授权用户。(2)~(5)的多重度也显而易见，(2)为 1，(3)为 0..*，(4)为 1，(5)为 0..*。

【问题 3】

候选类的选择运用了良性依赖原则：不会在实际中造成危害的依赖关系，都是良性依赖。通过分析不难发现，本原则的核心思想是“务实”，很好地揭示了极限编程(Extreme Programming)中“简单设计”和“重构”的理论基础。本原则可以帮助我们抵御“面向对象



设计 5 大原则”以及设计模式的诱惑，以免陷入过度设计(Over-engineering)的尴尬境地，带来不必要的麻烦。候选类的删除运用了接口隔离原则(ISP)：不应该强迫客户依赖于他们不用的方法。接口属于客户，不属于它所在的类层次结构。通俗点说就是不要强迫客户使用他们不用的方法，否则客户就会面临由于这些不使用的方法的改变所带来的改变。

答案：

【问题1】

A1：网络用户。A2：授权用户。UC1：认证。UC2：更新信息。

(1)：extend。

【问题 2】

C1：InternetClient。C2：Customer。C3：RegisteredClient。C4：Administrator。

C5：CustomerList。

(2)：1。(3)：0..*。(4)：1。(5)：0..*。

【问题 3】

候选类的选择运用了良性依赖原则(不会在实际中造成危害的依赖关系，都是良性依赖)，候选类的删除运用了接口隔离原则(ISP)。

3.1.3 同步练习

1. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2012 年 11 月试题三)

【说明】

某城市的各国家公园周边建造了许多供游客租用的小木屋和营地，为此，该城市设置了一个中心售票处和若干个区域售票处。游客若想租用小木屋或营地，必须前往中心售票处进行预订并用现金支付全额费用，所有的预订操作全部由售票处的工作人员手工完成。现欲开发一信息系统，实现小木屋和营地的预订及管理功能，以取代手工操作。该系统的主要功能描述如下。

(1) 管理预订申请。游客可以前往任何一个售票处提出预订申请，系统对来自各个售票处的预订申请进行统一管理。

(2) 预订。预订操作包含登记游客预订信息、计算租赁费用、付费等步骤。

(3) 支付管理。游客付费时可以选择现金和信用卡付款两种方式。使用信用卡支付可以享受 3%的折扣，使用现金支付没有折扣。

(4) 游客取消预订。预订成功之后，游客可以在任何时间取消预订，但需支付赔偿金，剩余部分则退还给游客。赔偿金的计算规则是，在预订入住时间之前的 48 小时内取消，支付租赁费用 10%的赔偿金；在预订入住时间之后取消，则支付租赁费用 50%的赔偿金。

(5) 自动取消预订。如果遇到恶劣天气(如暴雨、山洪等)，系统会自动取消所有的预订，发布取消预订消息，全额退款。

(6) 信息查询。售票处工作人员查询小木屋和营地的预订和使用情况，以判断是否能够批准游客的预订申请。

现采用面向对象方法开发上述系统，得到如表 3-5 所示的用例列表和表 3-6 所示的类列

表，对应的用例图和类图分别如图 3-15 和图 3-16 所示。

【问题 1】(6 分)

根据说明中的描述与表 3-5，给出图 3-15 中 UC1~UC6 处所对应的用例名称。

表 3-5　用例列表

用 例 名	说　明	用 例 名	说　明
ManageInquiries	管理预订申请	ManageCashPayment	现金支付
MakeReservation	预订	ManageCrCardPayment	信用卡支付
ManagePayment	支付管理	GetDiscount	计算付款折扣
CancelReservation	游客取消预订	AutoCancelReservation	系统自动取消预订
CheckAvailability	信息查询	CalculateRefund	计算取消预订的赔偿金
PublishMessage	发布取消预订信息		

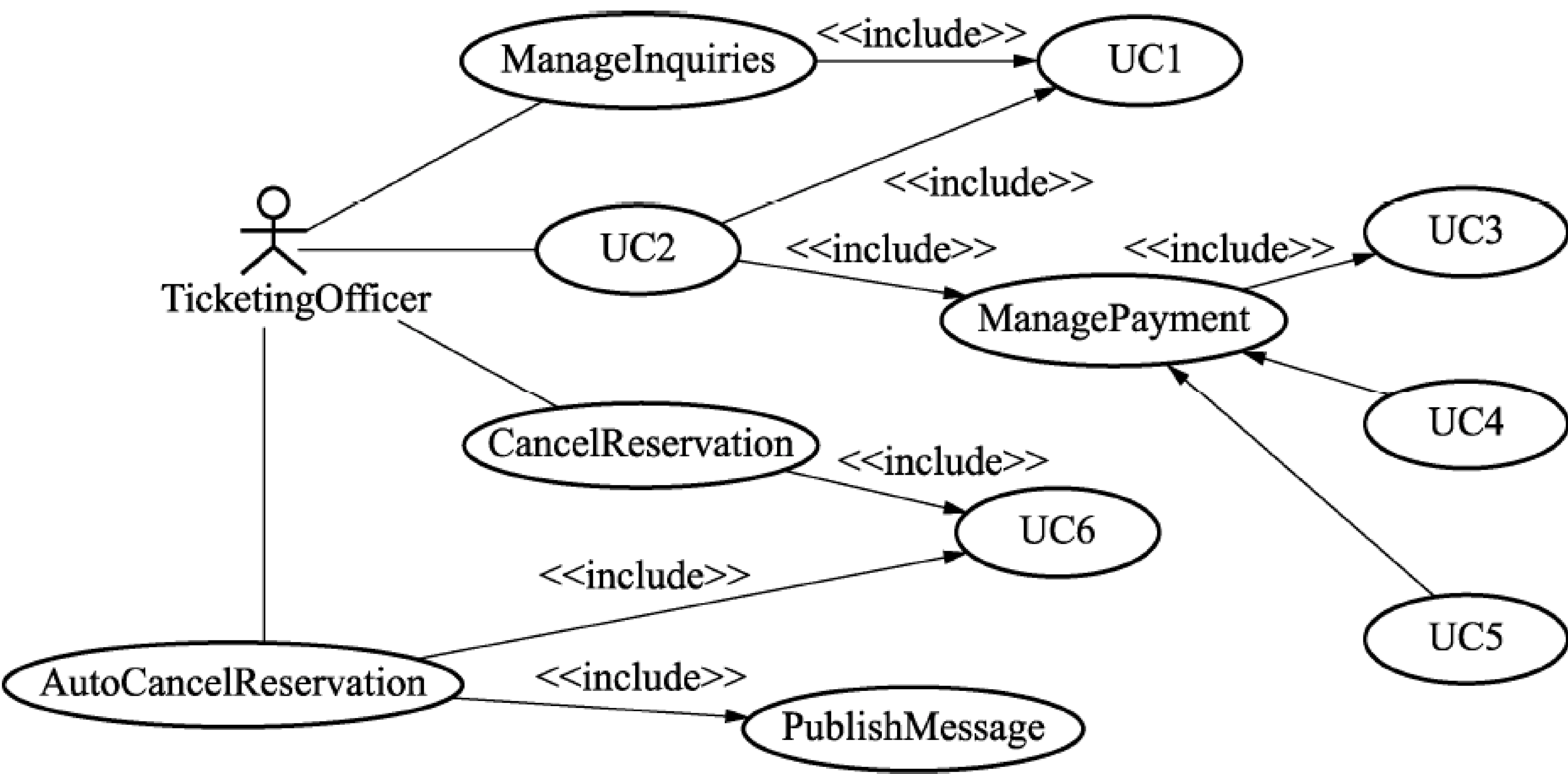


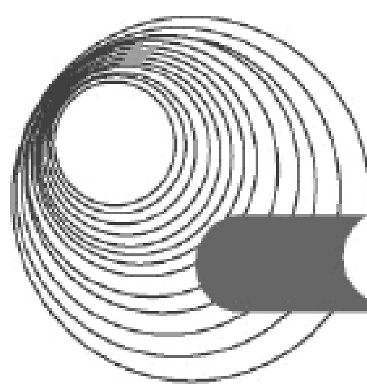
图 3-15　用例图

【问题 2】(7 分)

根据说明中的描述与表 3-6，给出图 3-16 中 C1~C7 处所对应的类名。

表 3-6　类列表

类　名	说　明	类　名	说　明
NationalPark	国家公园	Customer	游客
Reservation	预订申请	ReservationItem	预订申请内容
TicketingOfficer	售票处	CampSite	营地
Bungalow	小木屋	Payment	付款
Discount	付款折扣	CashPayment	现金支付
CreditCardPayment	信用卡支付	Rate	租赁费用



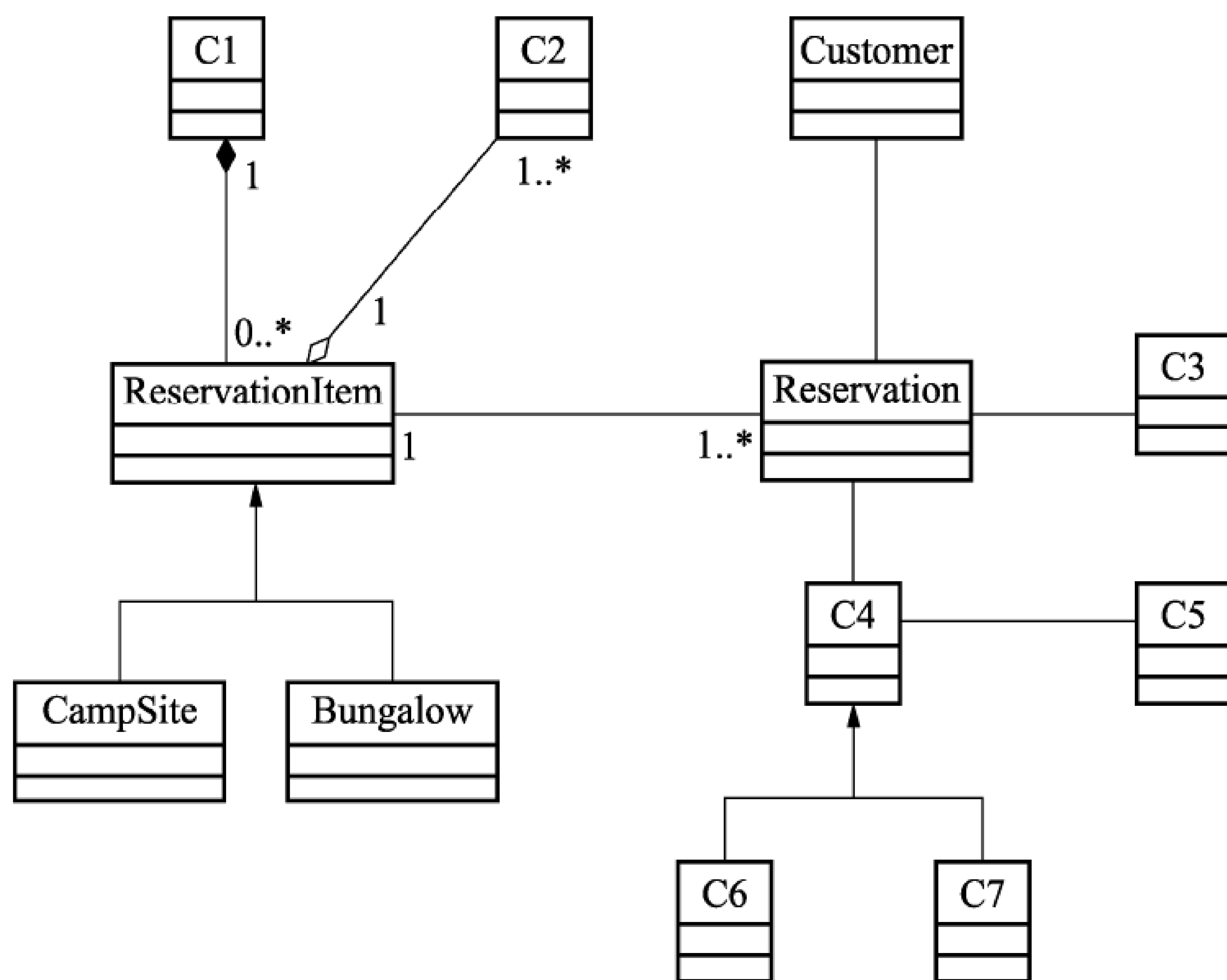


图 3-16　类图

【问题 3】(2 分)

对于某些需求量非常大的小木屋或营地，说明中功能(4)的赔偿金计算规则不足以弥补取消预订所带来的损失。如果要根据预订的时段以及所预订场地的需求量设计不同层次的赔偿金计算规则，需要对图 3-16 进行怎样的修改？(请用文字说明)

2. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2012 年 5 月试题三)

【说明】

某网上购物平台的主要功能如下。

(1) 创建订单。顾客(Customer)在线创建订单(Order)，主要操作是向订单中添加项目、从订单中删除项目。订单中应列出所订购的商品(Product)及其数量(quantities)。

(2) 提交订单。订单通过网络来提交。在提交订单时，顾客需要提供其姓名(name)、收货地址(address)以及付款方式(form of payment)(预付卡、信用卡或者现金)。为了制订送货计划以及安排送货车辆，系统必须确定订单量(volume)。除此之外，还必须记录每种商品的名称(name)、造价(cost price)、售价(sale price)以及单件商品的包装体积(cubic volume)。

(3) 处理订单。订单处理人员接收来自系统的订单；根据订单的内容安排配货，制订送货计划。在送货计划中不仅要指明发货日期(delivery date)，还要记录每个订单的限时发送要求(Delivery Time Window)。

(4) 派单。订单处理人员将已配好货的订单转交给派送人员。

(5) 送货/收货。派送人员将货物送到顾客指定的收货地址。当顾客收货时，需要在运货单(delivery slip)上签收。签收后的运货单最终需交还给订单处理人员。

(6) 收货确认。当订单处理人员收到签收过的运货单后，会和顾客进行一次再确认。

现采用面向对象方法开发上述系统，得到如图 3-17 所示的用例图和如图 3-18 所示的类图。

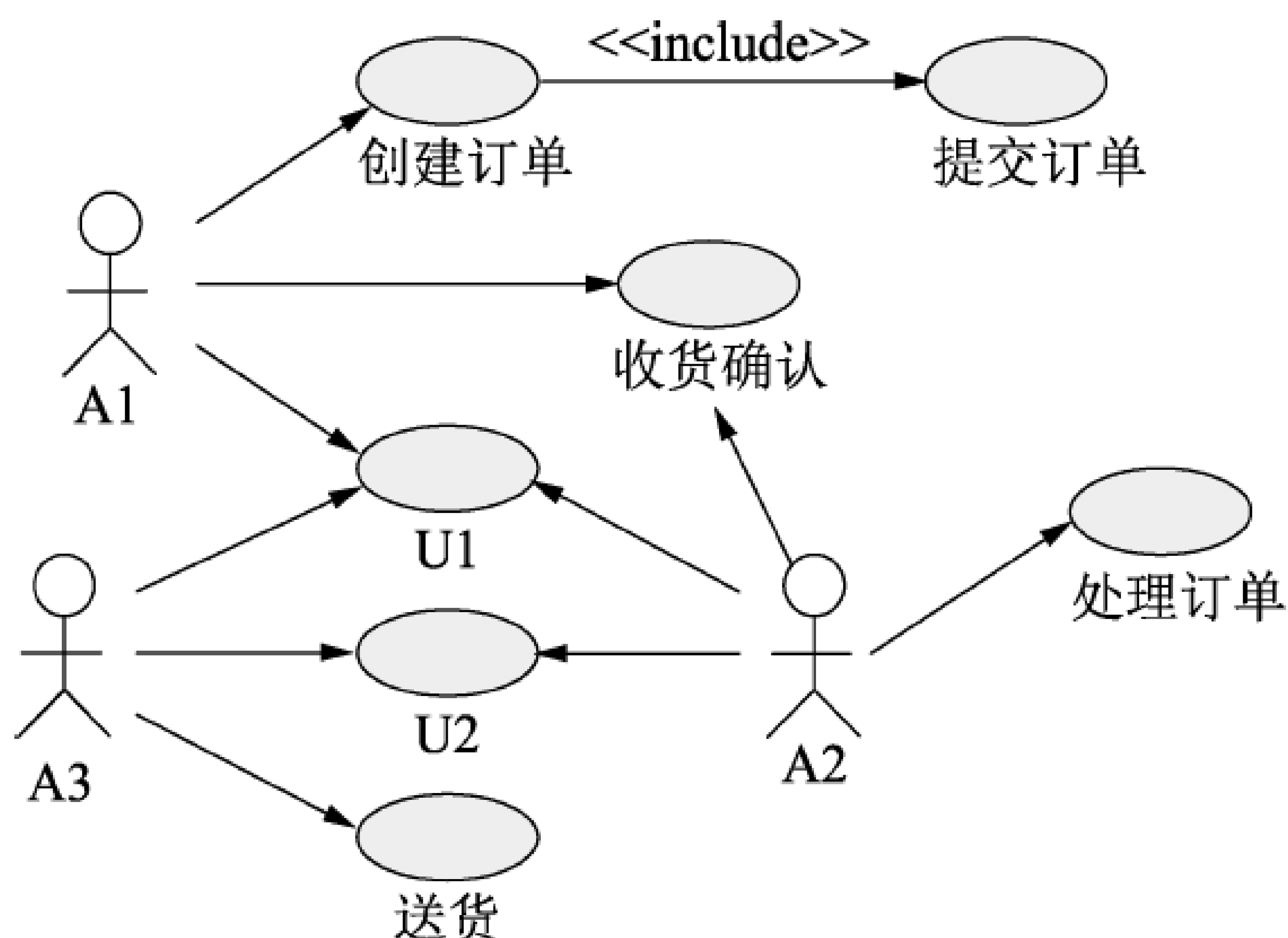


图 3-17　用例图

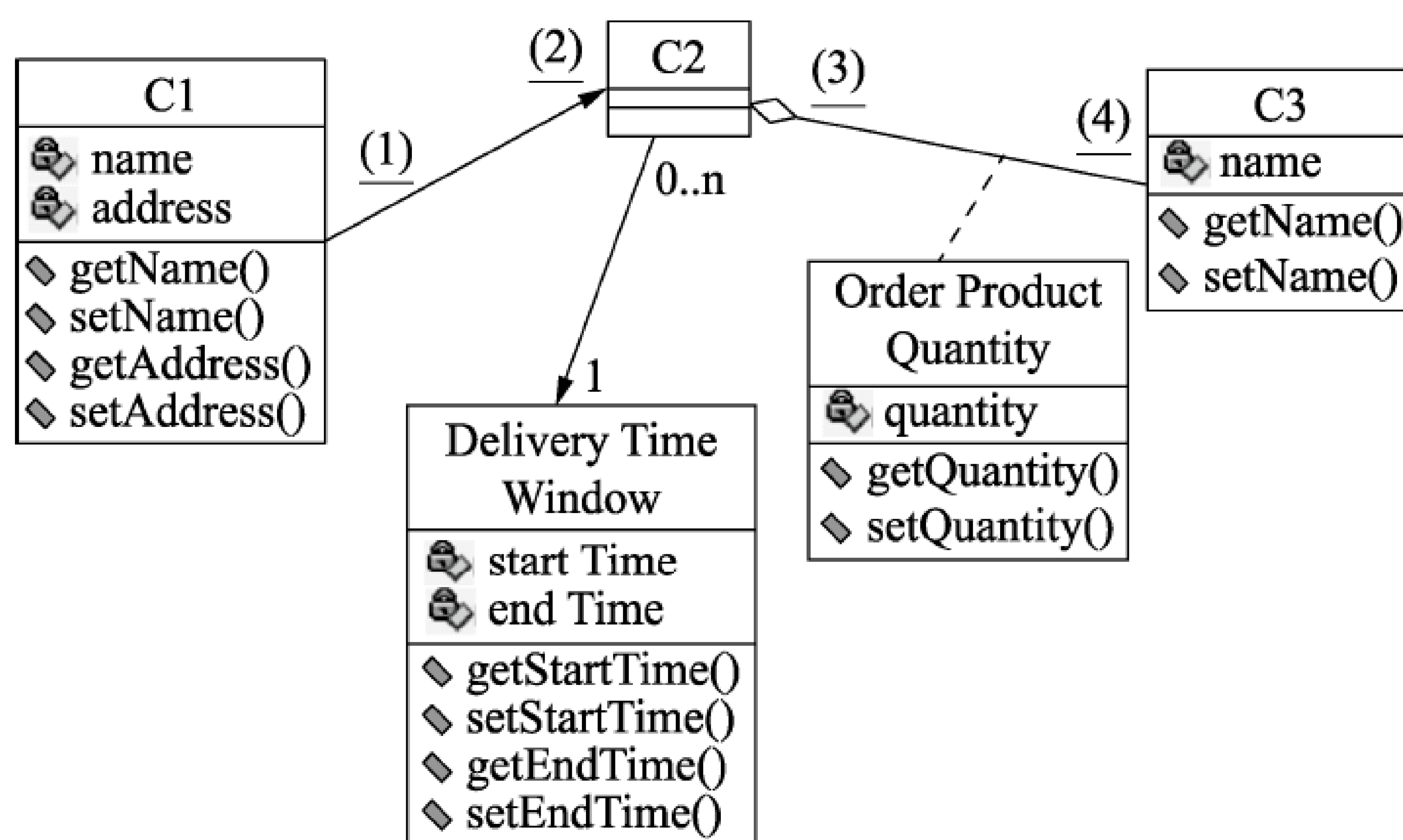


图 3-18　类图

【问题 1】(5 分)

根据说明中的描述，给出图 3-17 中 A1~A3 所对应的参与者名称和 U1、U2 处所对应的用例名称。

【问题 2】(7 分)

根据说明中的描述，给出图 3-18 中 C1~C3 所对应的类名以及(1)~(4)处所对应的多重度(类名使用说明中给出的英文词汇)。

【问题 3】(3 分)

根据说明中的描述，将类 C2 和 C3 的属性补充完整(属性名使用说明中给出的英文词汇)。

3. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2011 年 11 月试题三)

【说明】

Pay&Drive(开多少付多少)系统能够根据驾驶里程自动计算应付的费用。

系统中存储了特定区域道路交通网的信息。道路交通网由若干个路段(Road Segment)构成，每个路段由两个地理位置坐标点(Node)标定，其里程数(Distance)是已知的。在某些地

理位置坐标点上安装了访问控制(Access Control)设备，可以自动扫描行驶卡(Card)。行程(Trajectory)由一组连续的路段构成。行程的起点(Entry)和终点(Exit)都装有访问控制设备。

系统提供了 3 种行驶卡。常规卡(Regular Card)有效期(Valid Period)为一年，可以在整个道路交通网内使用。季卡(Season Card)有效期为三个月，可以在整个道路交通网内使用。单次卡(Minitrip Card)在指定的行程内使用，且只能使用一次。其中，季卡和单次卡都是预付卡(Prepaid Card)，需要客户(Customer)预存一定的费用。

系统的主要功能有客户注册、申请行驶卡、使用行驶卡行驶等。

使用常规卡行驶，在进入行程起点时，系统记录行程起点、进入时间(Date of Entry)等信息。在到达行程终点时，系统根据行驶的里程数和所持卡的里程单价(Unit Price)计算应付费用，并打印费用单(Invoice)。

季卡的使用流程与常规卡类似，但是不需要打印费用单，系统自动从卡中扣除应付费用。

单次卡的使用流程与季卡类似，但是还需要在行程的起点和终点上检查行驶路线是否符合该卡所规定的行驶路线。

现采用面向对象方法开发该系统，使用 UML 进行建模。构建出的用例图和类图分别如图 3-19 和图 3-20 所示。

【问题 1】(4 分)

根据说明中的描述，给出图 3-19 中的 U1 和 U2 所对应的用例，以及(1)所对应的关系。

【问题 2】(8 分)

根据说明中的描述，给出图 3-20 中缺少的 C1~C6 所对应的类名以及(2)~(3)处所对应的多重度(类名使用说明中给出的英文词汇)。

【问题 3】(3 分)

根据说明中的描述，给出 Road Segment、Trajectory 和 Card 所对应的类的关键属性(属性名使用说明中给出的英文词汇)。

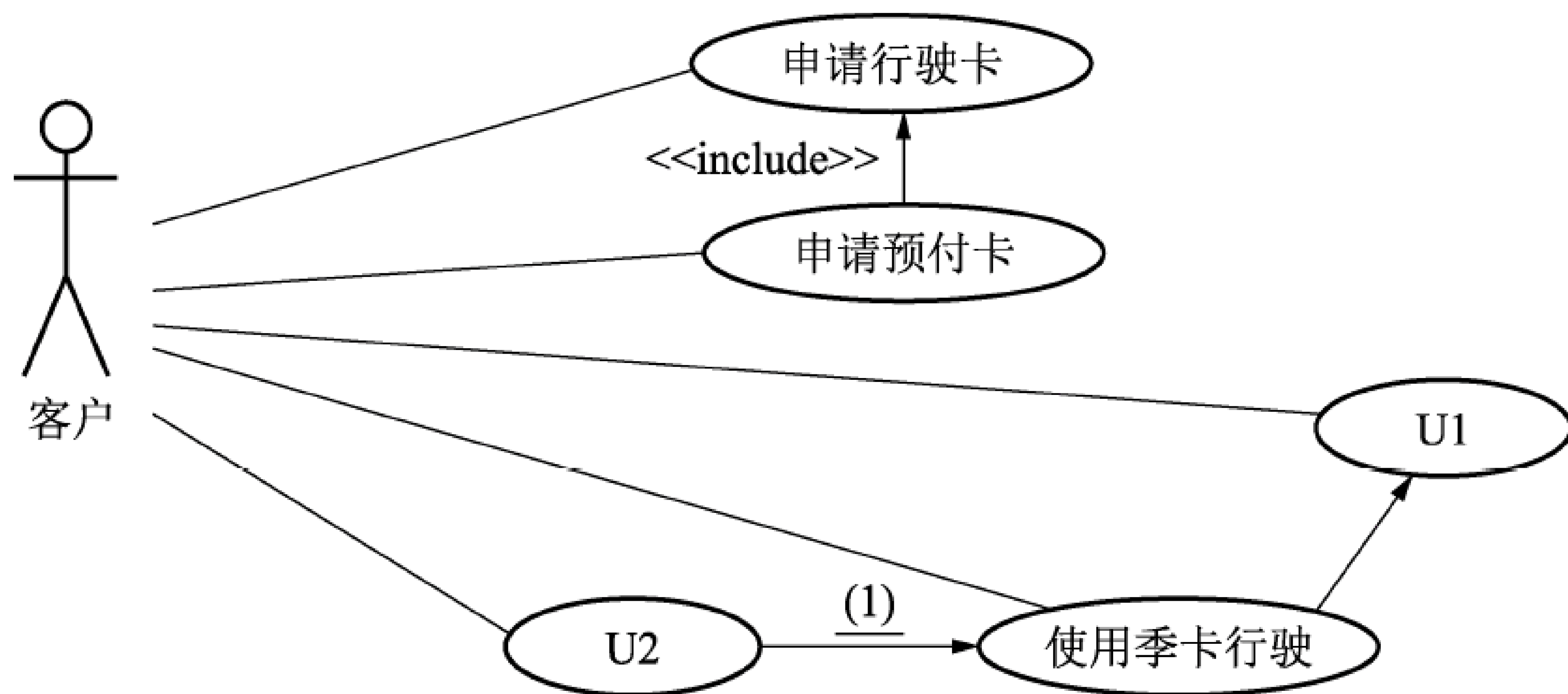


图 3-19　用例图

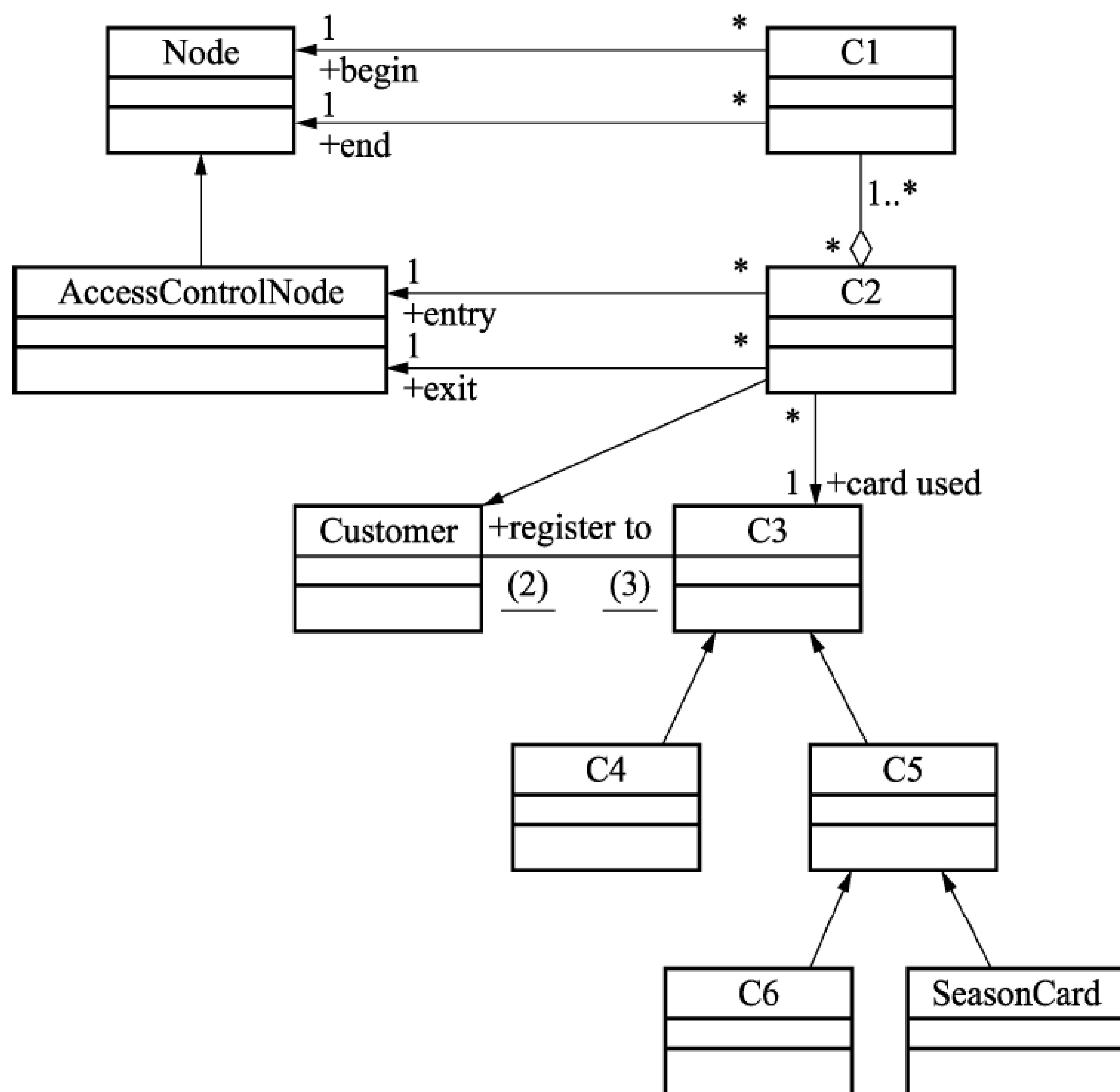


图 3-20　类图

4. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2011 年 5 月试题三)

【说明】

一个简单的图形编辑器提供给用户的基本操作包括创建图形、创建元素、选择元素以及删除图形。图形编辑器的组成及其基本功能描述如下。

(1) 图形由文本元素和图元元素构成，图元元素包括线条、矩形和椭圆。

(2) 图形显示在工作空间中，一次只能显示一张图形(即当前图形，current)。

(3) 编辑器提供了两种操作图形的工具：选择工具和创建工具。对图形进行操作时，一次只能使用一种工具(即当前活动工具，active)。

① 创建工具用于创建文本元素和图形元素。

② 对于显示在工作空间中的图形，使用选择工具能够选定其中所包含的元素，可以选择一个元素，也可以同时选择多个元素。被选择的元素成为当前选中元素(selected)。

③ 每种元素都具有相应的控制点。拖曳选定元素的控制点，可以移动元素或者调整元素的大小。

现采用面向对象方法开发该图形编辑器，使用 UML 进行建模。构建出的用例图和类图分别如图 3-21 和图 3-22 所示。

【问题 1】(4 分)

根据说明中的描述，给出图 3-21 中 U1 和 U2 所对应的用例，以及(1)和(2)处所对应的关系。

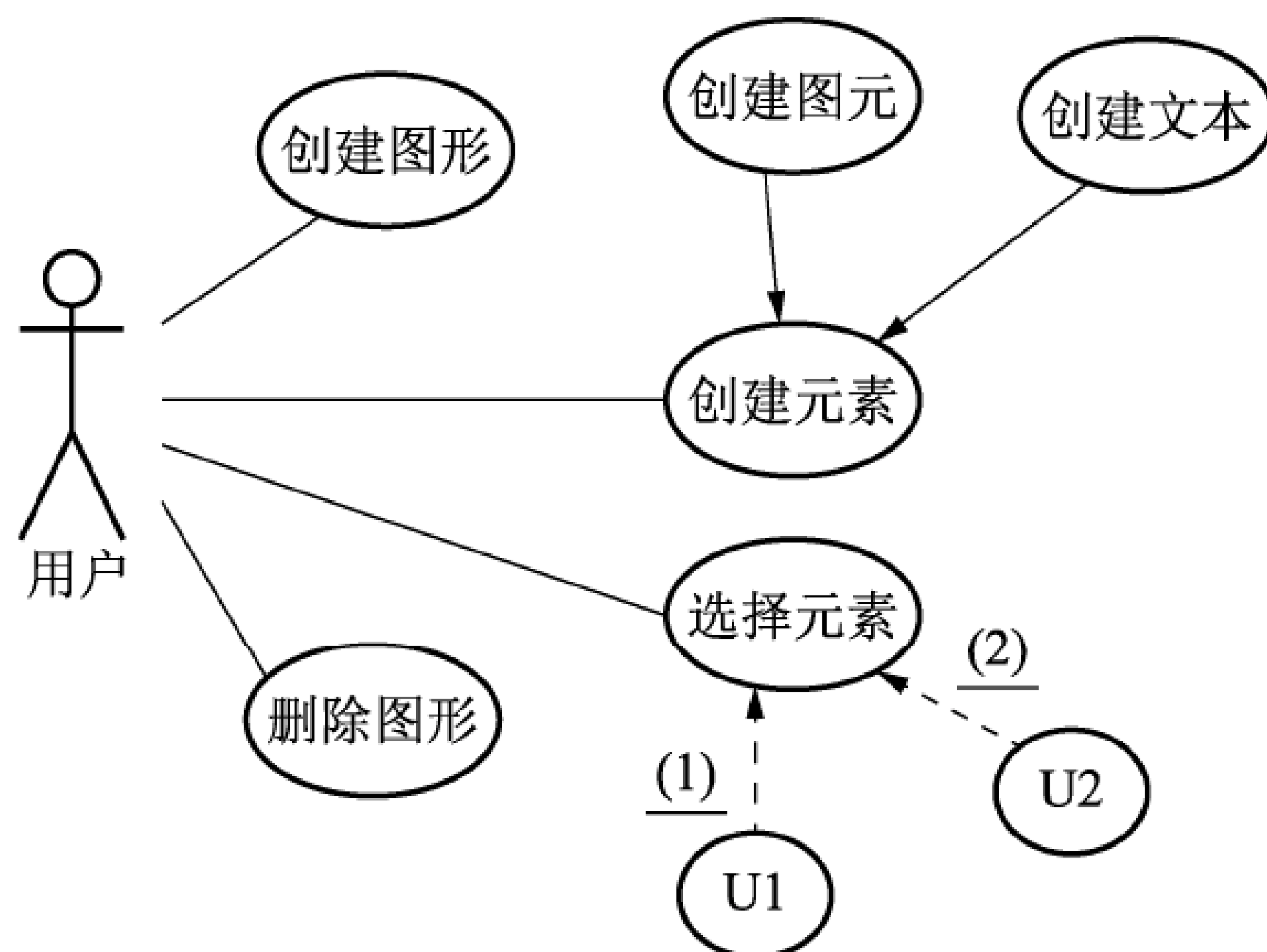


图 3-21　用例图

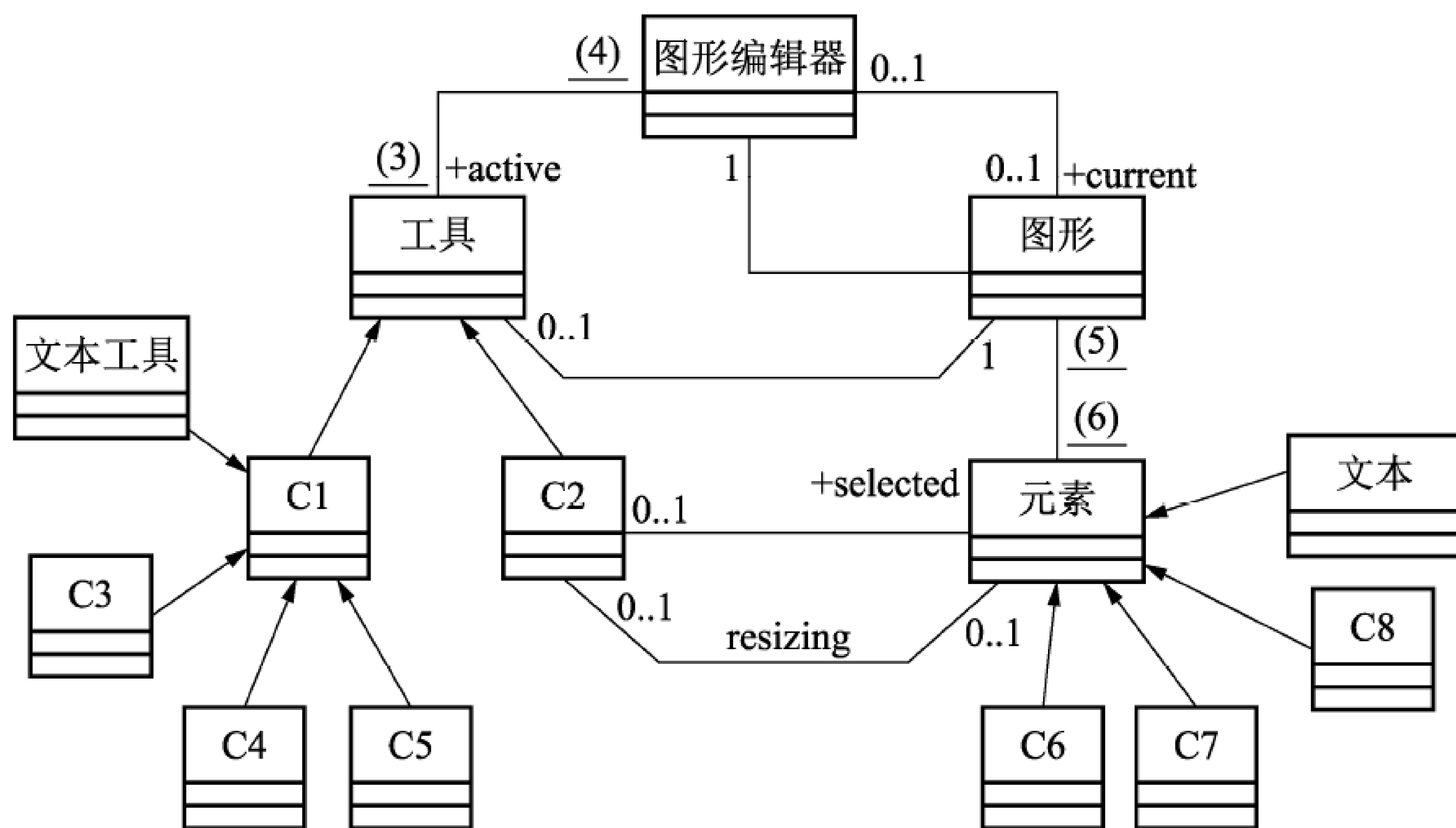


图 3-22　类图

【问题 2】(8 分)

根据说明中的描述，给出图 3-22 中缺少的 C1~C8 所对应的类名以及(3)~(6)处所对应的多重度。

【问题 3】(3 分)

图 3-22 中的类图采用了桥接(Bridge)设计模式，请说明该模式的内涵。

5. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2010 年 11 月试题三)

【说明】

某网上药店允许顾客凭借医生开具的处方，通过网络在该药店购买处方上的药品。该网上药店的基本功能描述如下。

(1) 注册。顾客在买药之前，必须先在网上药店注册。注册过程中需填写顾客资料以及付款方式(信用卡或者支付宝账户)。此外顾客必须与药店签订一份授权协议书，授权药店可以向其医生确认处方的真伪。

(2) 登录。已经注册的顾客可以登录到网上药房购买药品。如果是没有注册的顾客，系统将拒绝其登录。

(3) 录入及提交处方。登录成功后，顾客按照“处方录入界面”显示的信息，填写开具处方的医生信息以及处方上的药品信息。填写完成后，提交该处方。

(4) 验证处方。对于已经提交的处方(系统将其状态设置为“处方已提交”)，其验证过程如下。

① 核实医生信息。如果医生信息不正确，该处方的状态被设置为“医生信息无效”，并取消这个处方的购买请求；如果医生信息是正确的，系统给该医生发送处方确认请求，并将处方状态修改为“审核中”。

② 如果医生回复处方无效，系统会取消处方，并将处方状态设置为“无效处方”。如果医生没有在 7 天内给出确认答复，系统也会取消处方，并将处方状态设置为“无法审核”。

③ 如果医生在 7 天内给出了确认答复，该处方的状态被修改为“准许付款”。

系统取消所有未通过验证的处方，并自动发送一封电子邮件给顾客，通知顾客处方被取消以及取消的原因。

(5) 对于通过验证的处方，系统自动计算药品的价格并邮寄药品给已经付款的顾客。

该网上药店采用面向对象的方法开发，使用 UML 进行建模。系统的类图如图 3-23 所示。

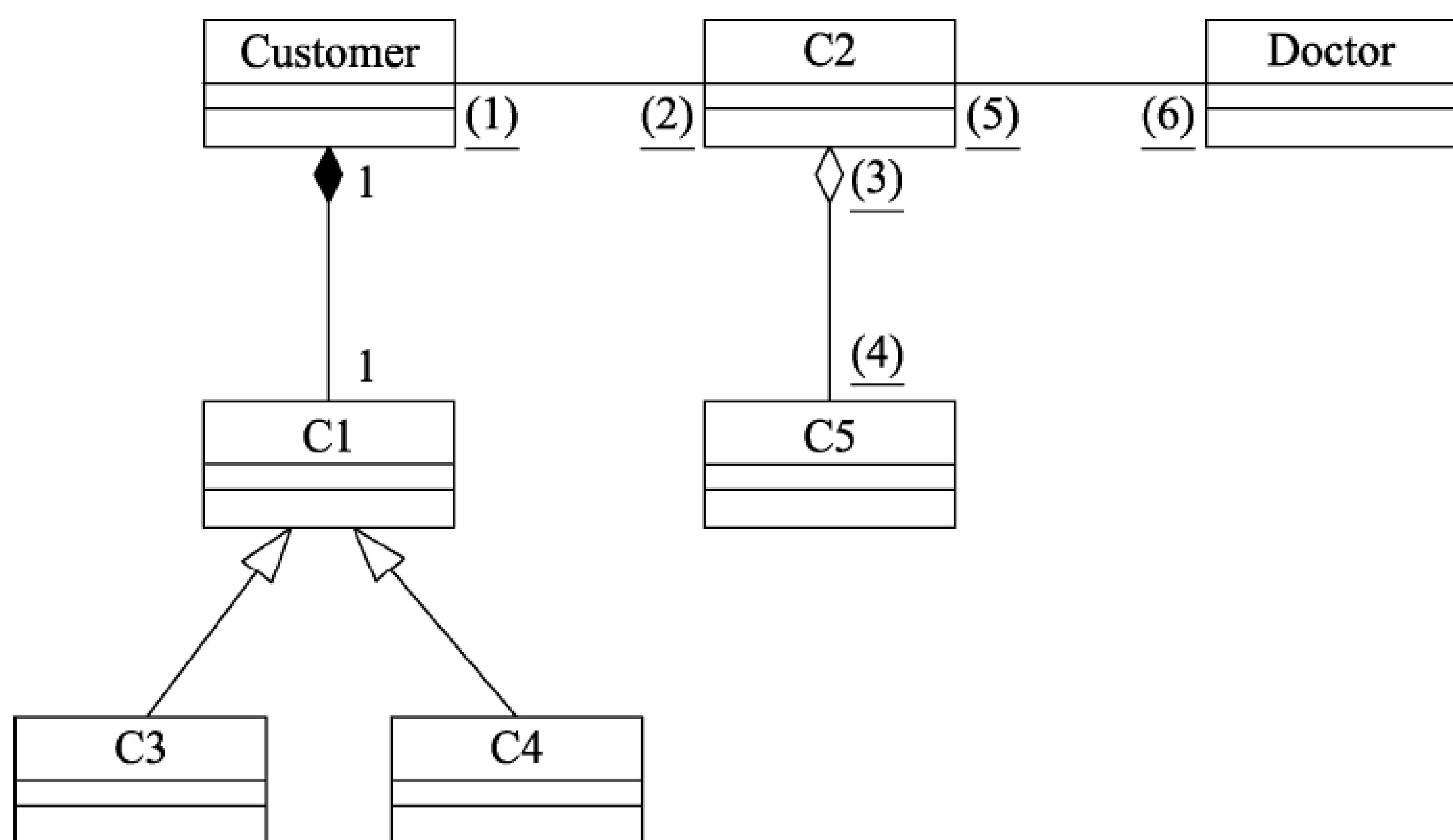


图 3-23　类图

【问题 1】(8 分)

根据说明中的描述，给出图 3-23 中缺少的 C1~C5 所对应的类名以及(1)~(6)处所对应的多重度。

【问题 2】(4 分)

图 3-24 给出了“处方”的部分状态图。根据说明中的描述，给出图 3-24 中缺少的 Sl~S4 所对应的状态名以及(7)~(10)处所对应的迁移(transition)名。

【问题 3】(3 分)

图 3-23 中的符号“◆”和“◇”在 UML 中分别表示类和对象之间的哪两种关系？两者之间的区别是什么？

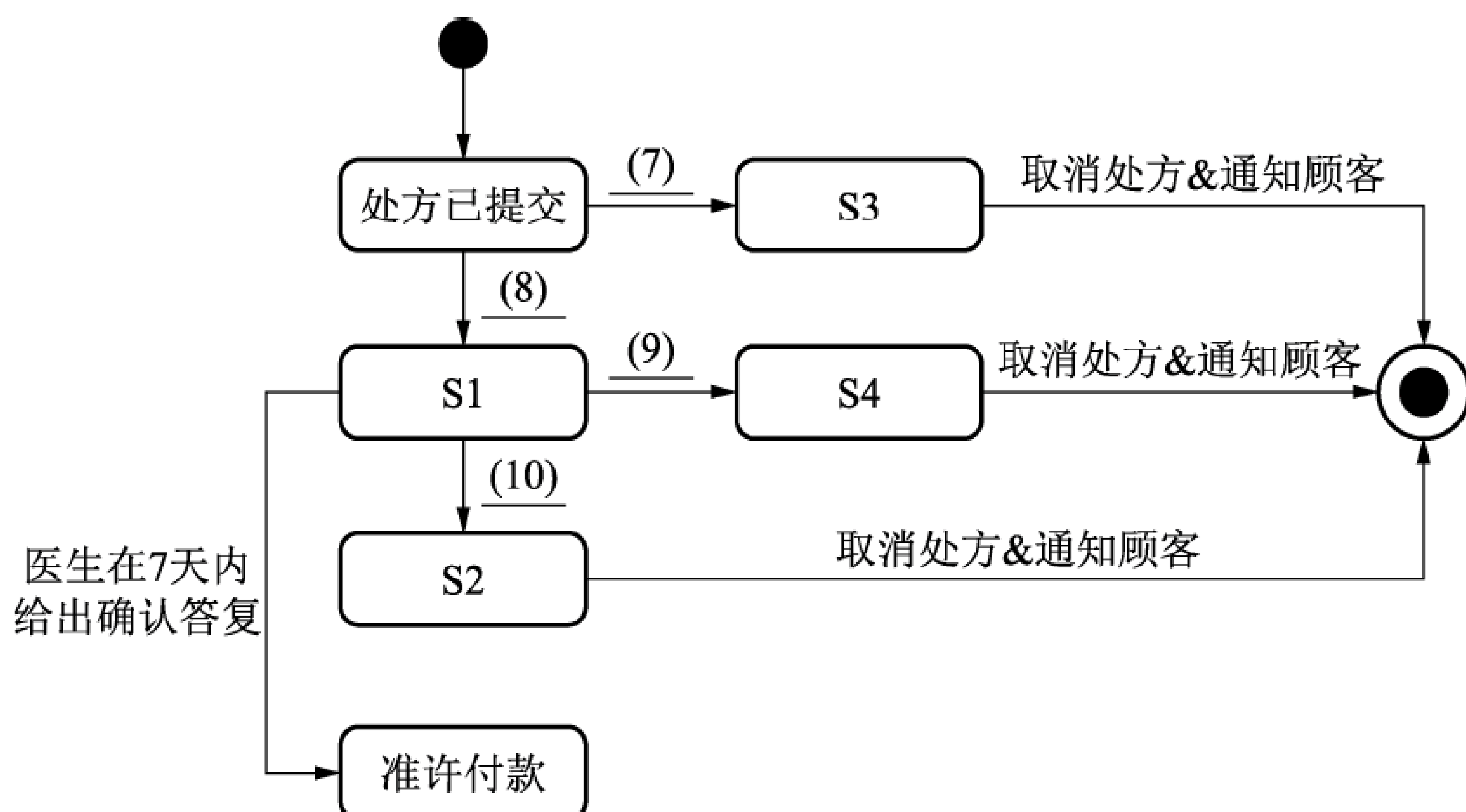


图 3-24　状态图

6. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2010 年 5 月试题三)

【说明】

某运输公司决定为新的售票机开发车票销售的控制软件。图 3-25 给出了售票机的面板示意图以及相关的控制部件。

售票机相关部件的作用如下所述。

(1) 目的地键盘用来输入行程目的地的代码(如 200 表示总站)。

(2) 乘客可以通过车票键盘选择车票种类(单程票、多次往返票和座席种类)。

(3) 继续/取消键盘上的“取消”按钮用于取消购票过程，“继续”按钮允许乘客连续购买多张票。

(4) 显示屏显示所有的系统输出和用户提示信息。

(5) 插卡口接受 MCard(现金卡)，硬币口和纸币槽接受现金。

(6) 打印机用于输出车票。

假设乘客总是支付恰好需要的金额而无须找零，售票机的维护工作(取回现金、放入空白车票等)由服务技术人员完成。

系统采用面向对象方法开发，使用 UML 进行建模。系统的顶层用例图和类图分别如图 3-26 和图 3-27 所示。

【问题 1】(5 分)

根据说明中的描述，给出图 3-26 中 A1 和 A2 所对应的参与者、U1 所对应的用例，以及(1)、(2)处所对应的关系。

【问题 2】(7 分)

根据说明中的描述，给出图 3-27 中缺少的 C1~C4 所对应的类名，以及(3)~(6)处所对应的多重度。

【问题 3】(3 分)

图 3-27 所示的类图设计采用了中介者(Mediator)设计模式，请说明该模式的内涵。

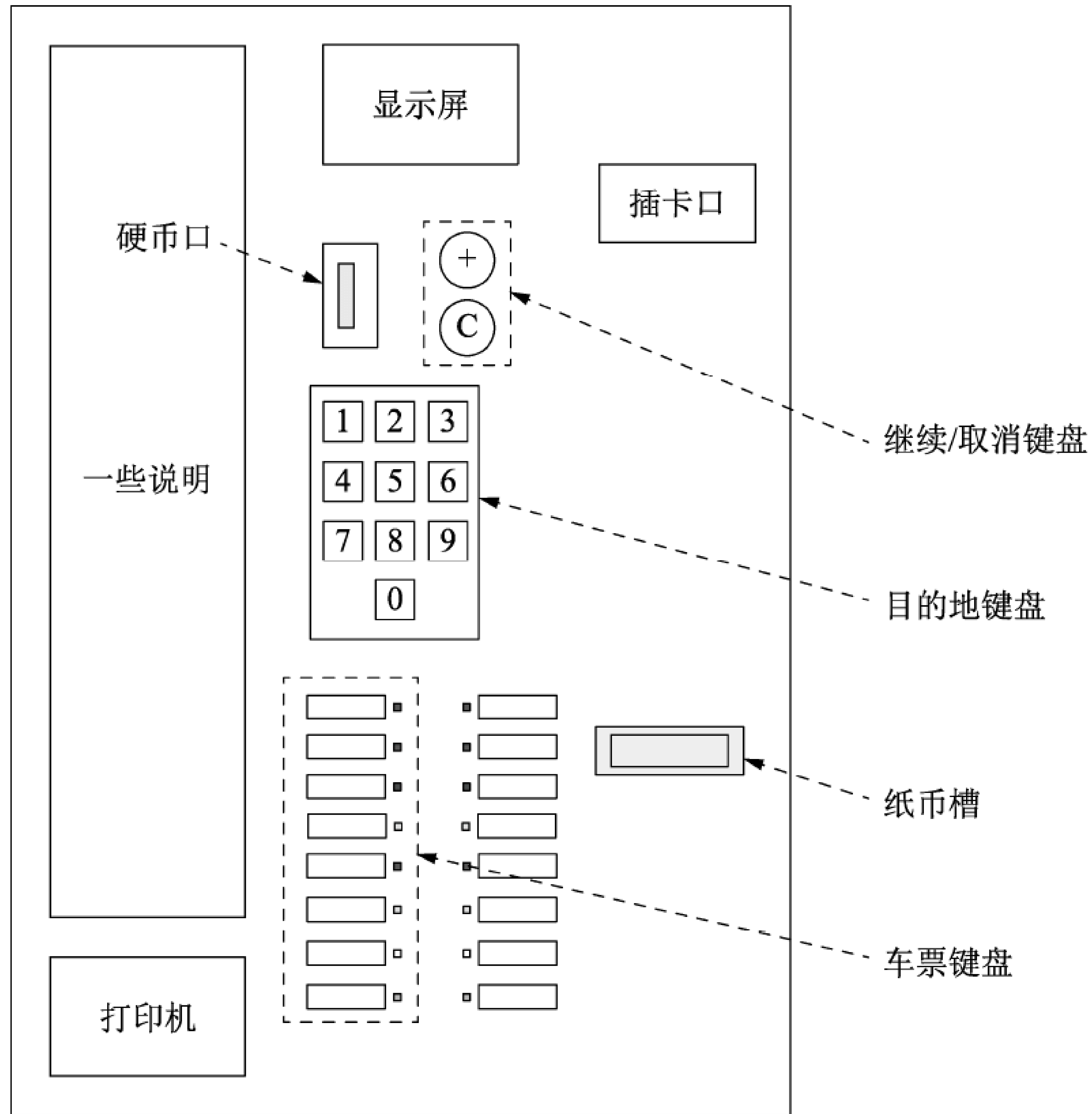


图 3-25　售票机面板示意图

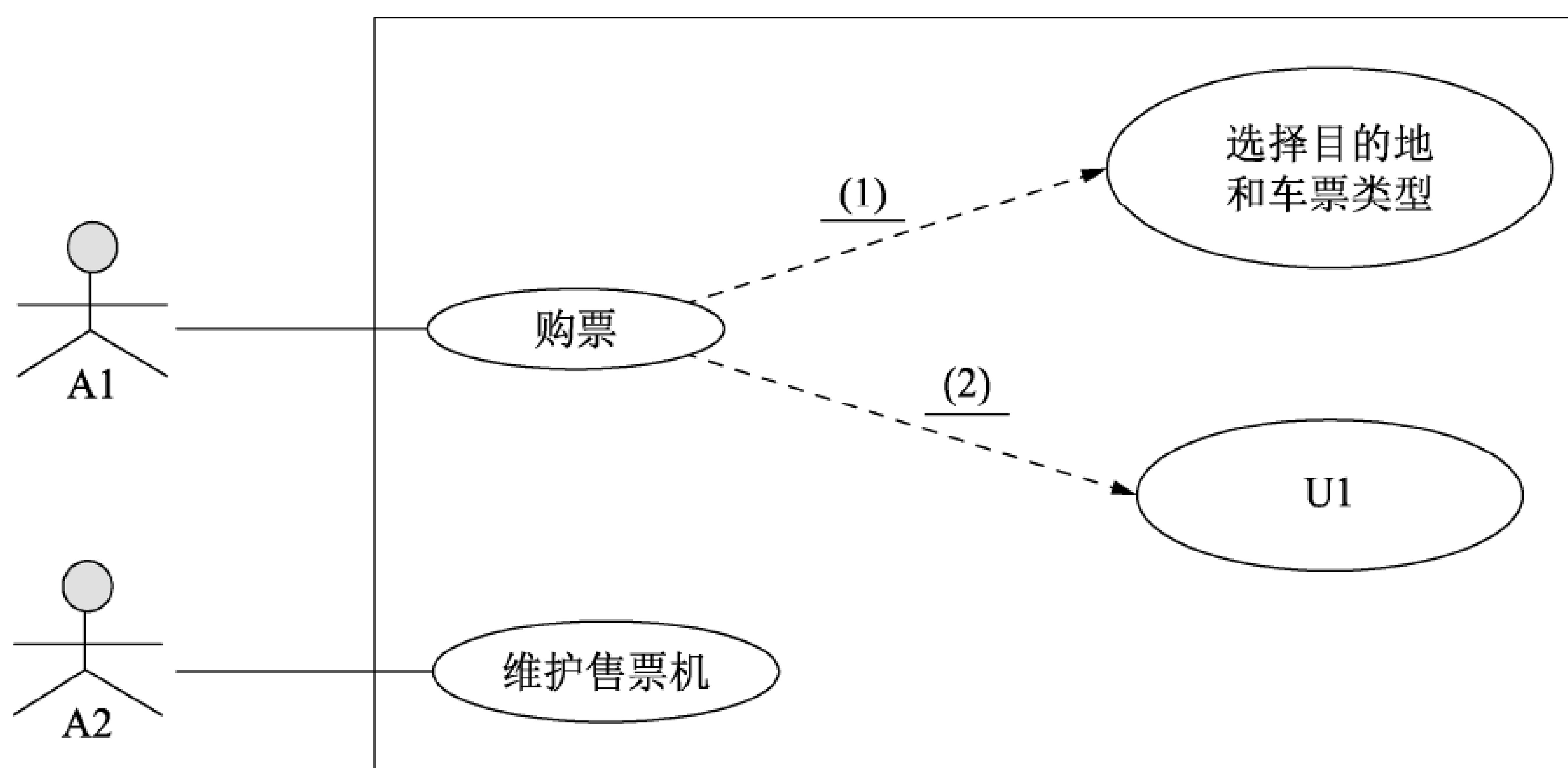


图 3-26　顶层用例图

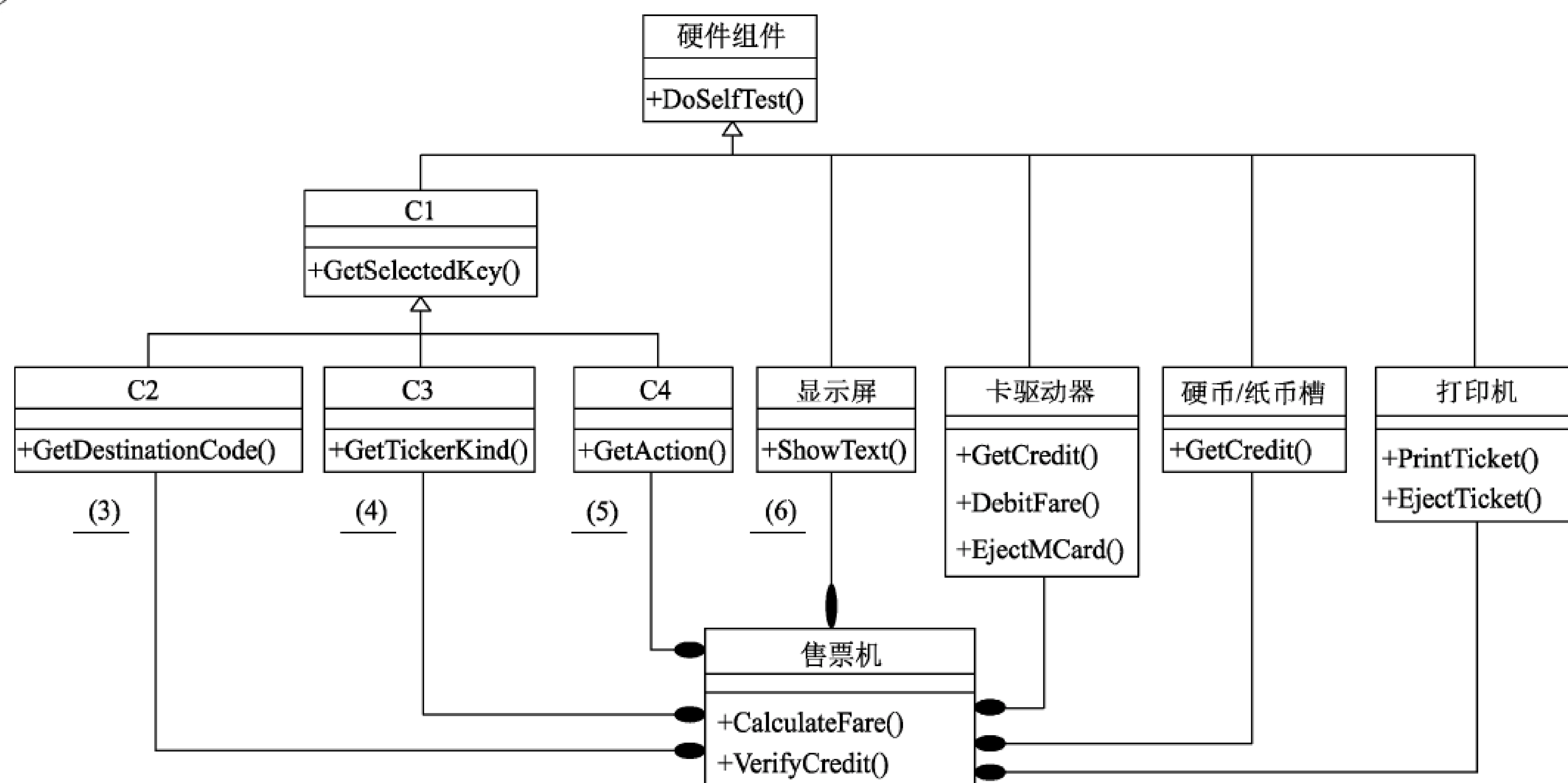


图 3-27　类图

7. 阅读下列说明和 UML 图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2009 年 11 月试题三)

【说明】

某企业餐厅为了方便员工用餐，开发了一个订餐系统(Cafeteria Ordering System，COS)，企业员工可通过企业内联网使用该系统。

企业的任何员工都可以查看菜单和当日特价。

系统的顾客是注册到系统的员工，可以订餐(如果未登录则需先登录)、注册工资支付、预约规律的订餐，在特殊情况下可以覆盖预订。

餐厅员工是特殊顾客，可以进行备餐、生成付费请求和请求送餐，其中对于注册工资支付的顾客生成付费请求并发送给工资系统。

菜单管理员是餐厅特定员工，可以管理菜单。

送餐员可以打印送餐说明、记录送餐信息(如送餐时间)及记录收费(对于没有注册工资支付的顾客，由送餐员收取现金后记录)。

顾客的订餐过程如下。

(1) 顾客请求查看菜单。

(2) 系统显示菜单和当日特价。

(3) 顾客选菜。

(4) 系统显示订单和价格。

(5) 顾客确认订单。

(6) 系统显示可送餐时间。

(7) 顾客指定送餐时间、地点和支付方式。

(8) 系统确认接受订单，然后发送 E-mail 给顾客以确认订餐，同时发送相关订餐信息通知给餐厅员工。

系统采用面向对象方法开发，使用 UML 进行建模。系统的顶层用例图和一次订餐的活

动图初稿分别如图 3-28 和图 3-29 所示。

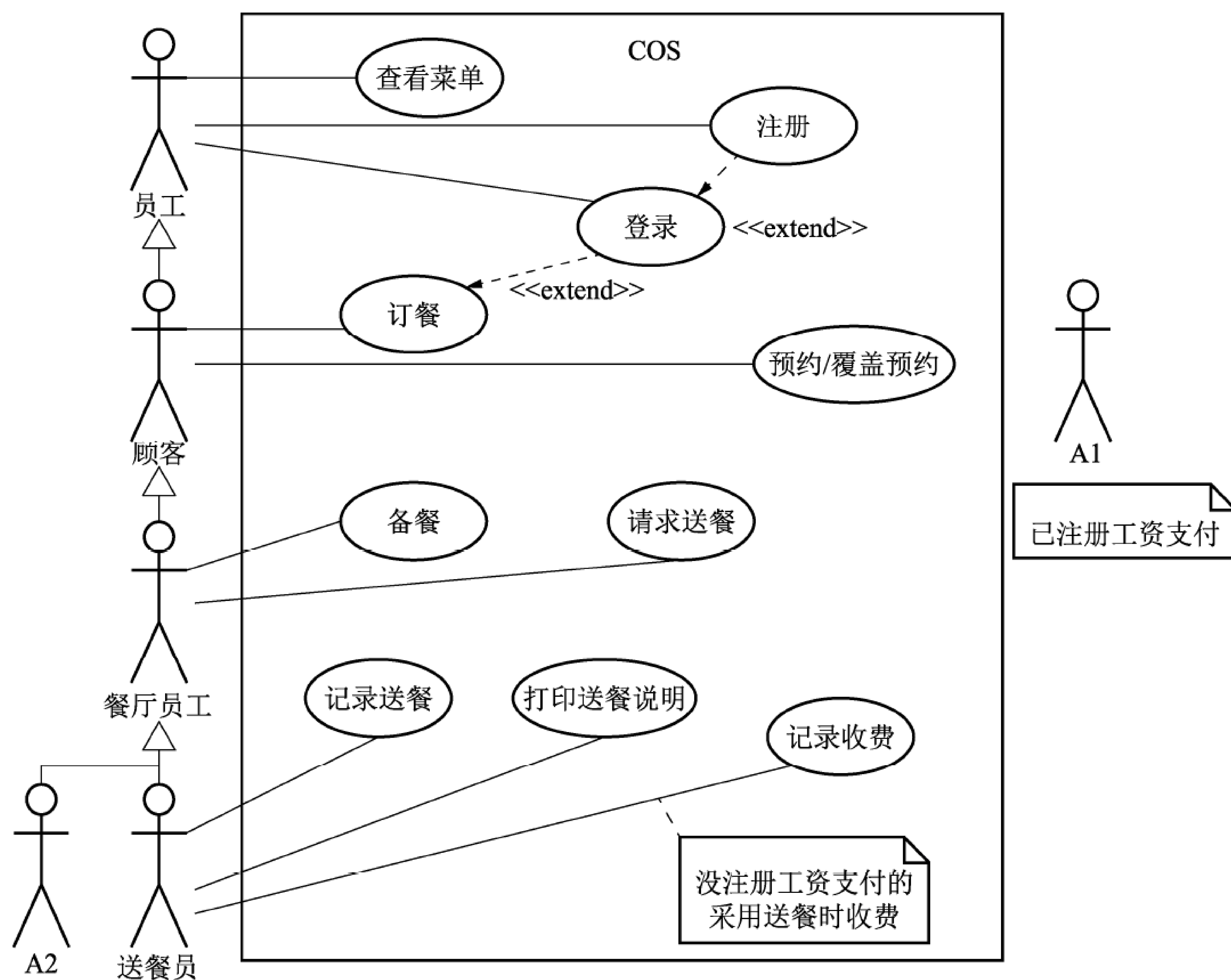


图 3-28　COS 系统顶层用例图

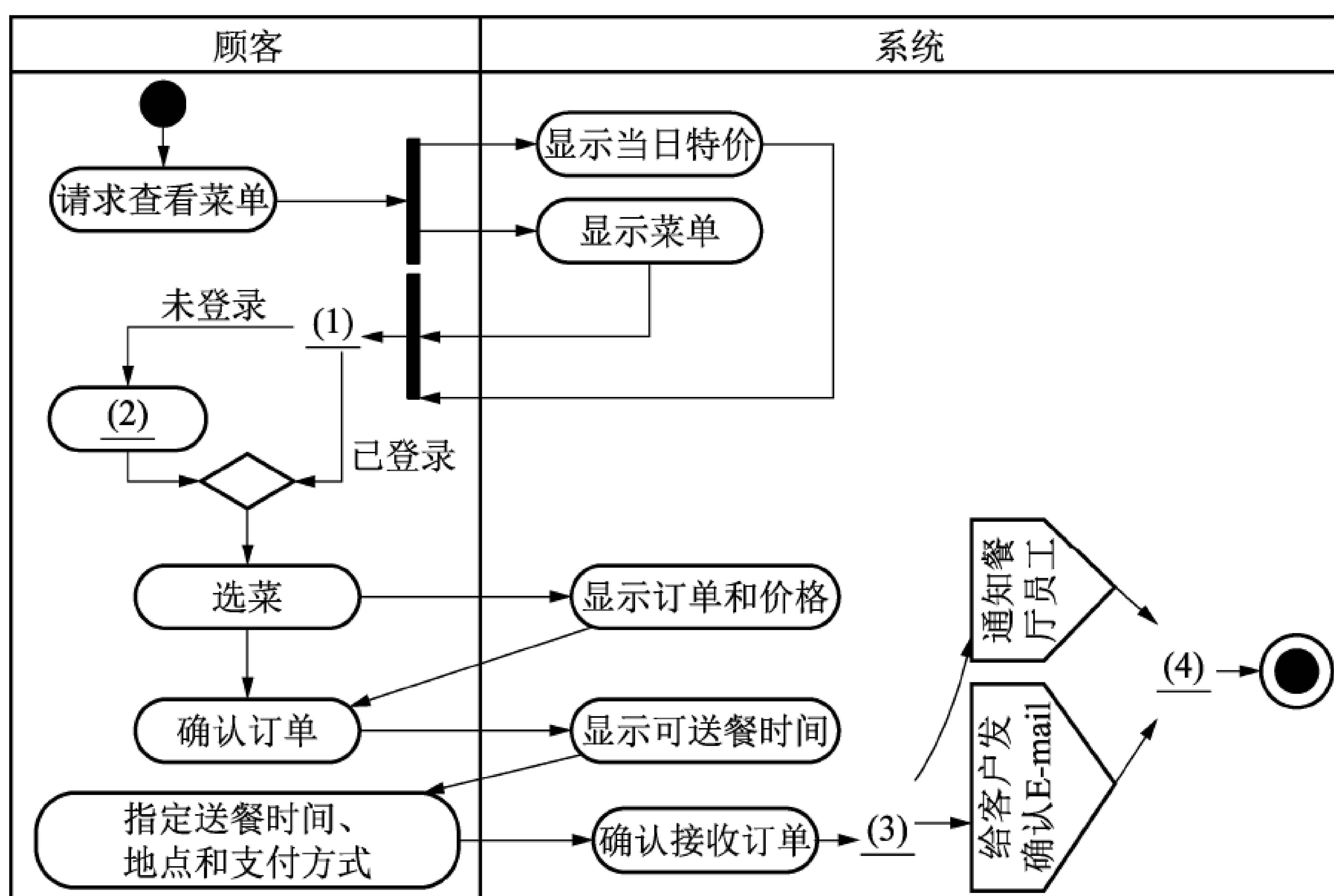


图 3-29　一次订餐的活动图

【问题 1】

根据说明中的描述，给出图 3-28 中 A1 和 A2 所对应的参与者。

【问题 2】

根据说明中的描述，给出图 3-28 中缺少的 4 个用例及其所对应的参与者。

【问题 3】

根据说明中的描述，给出图 3-29 中(1)~(4)处对应的活动名称或图形符号。

【问题 4】

指出图 3-28 中员工和顾客之间是什么关系，并解释该关系的内涵。

8. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2009 年 5 月试题三)

【说明】

某银行计划开发一个自动存提款机模拟系统(ATM System)。系统通过读卡器(Card Reader)读取 ATM 卡；系统与客户(Customer)的交互由客户控制台(Customer Console)实现；银行操作员(Operator)可控制系统启动(System Startup)和系统停止(System Shutdown)；系统通过网络和银行系统(Bank)实现通信。

当读卡器判断用户已将 ATM 卡插入后，创建会话(Session)。会话开始后，读卡器进行读卡，并要求客户输入个人验证码(PIN)。系统将卡号和个人验证码信息送到银行系统进行验证。验证通过后，客户可从菜单选择以下事务(Transaction)。

(1) 从 ATM 卡账户取款(Withdraw)。

(2) 向 ATM 卡账户存款(Deposit)。

(3) 进行转账(Transfer)。

(4) 查询(Inquire)ATM 卡账户信息。

一次会话可以包含多个事务，每个事务处理也会将卡号和个人验证码信息送到银行系统进行验证。若个人验证码错误，则转个人验证码错误处理(Invalid PIN Process)。每个事务完成后，客户可选择继续上述事务或退卡。选择退卡时，系统弹出 ATM 卡，会话结束。

系统采用面向对象方法开发，使用 UML 进行建模。系统的顶层用例图如图 3-30 所示，一次会话的序列图(不考虑验证)如图 3-31 所示。可能的消息名称列表如表 3-7 所示。

【问题 1】

根据说明中的描述，给出图 3-30 中 A1 和 A2 所对应的参与者、U1~U3 所对应的用例，以及该图中空(1)所对应的关系(U1~U3 的可选用例包括 Session、Transaction、Insert Card、Invalid PIN Process 和 Transfer)。

【问题 2】

根据说明中的描述，使用表 3-7 中的英文名称，给出图 3-31 中 6~9 对应的消息。

【问题 3】

解释图 3-30 中用例 U3 和用例 Withdraw、Deposit、Transfer、Inquire 等 4 个用例之间的关系及其内涵。

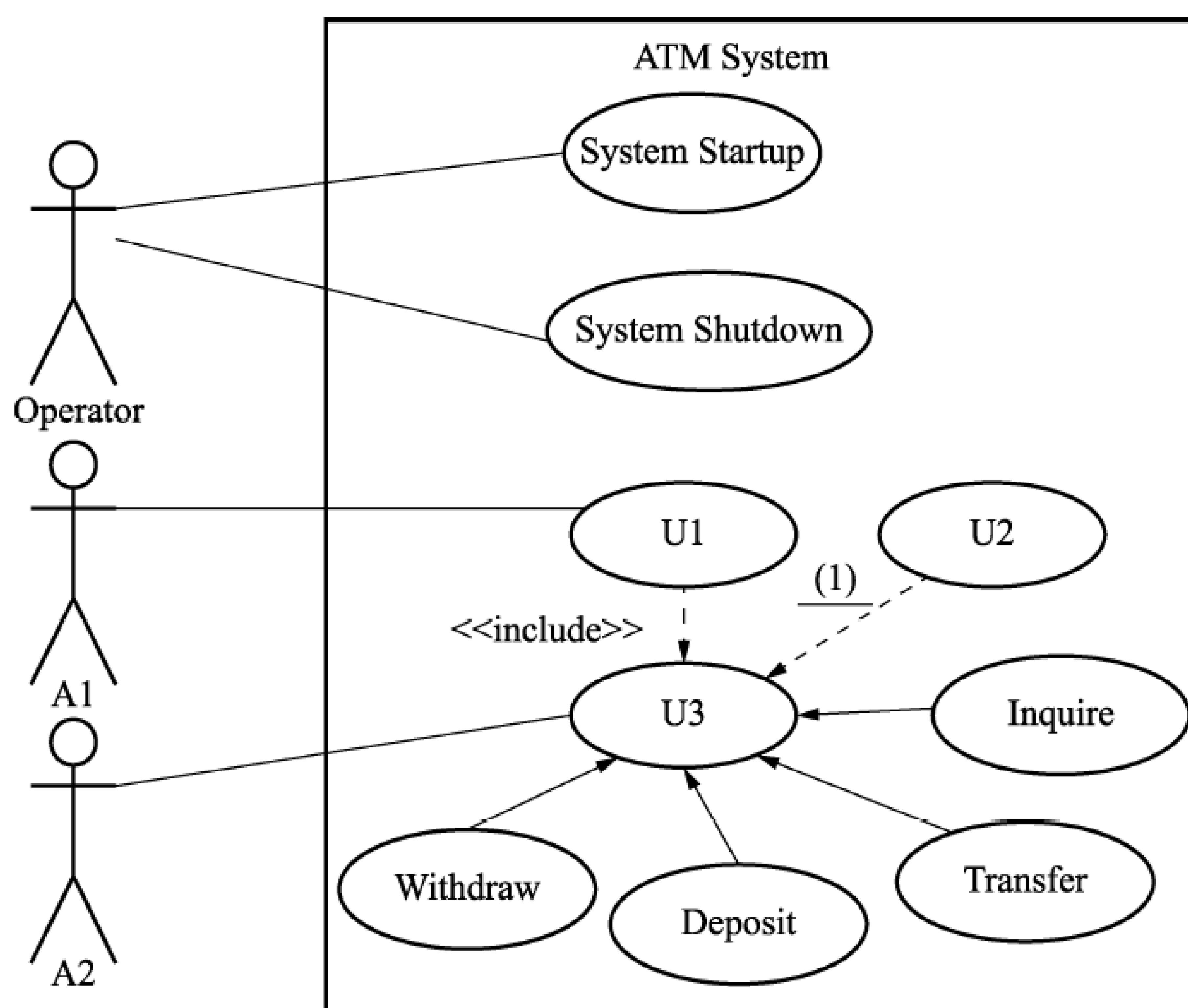


图 3-30　ATM 系统顶层用例图

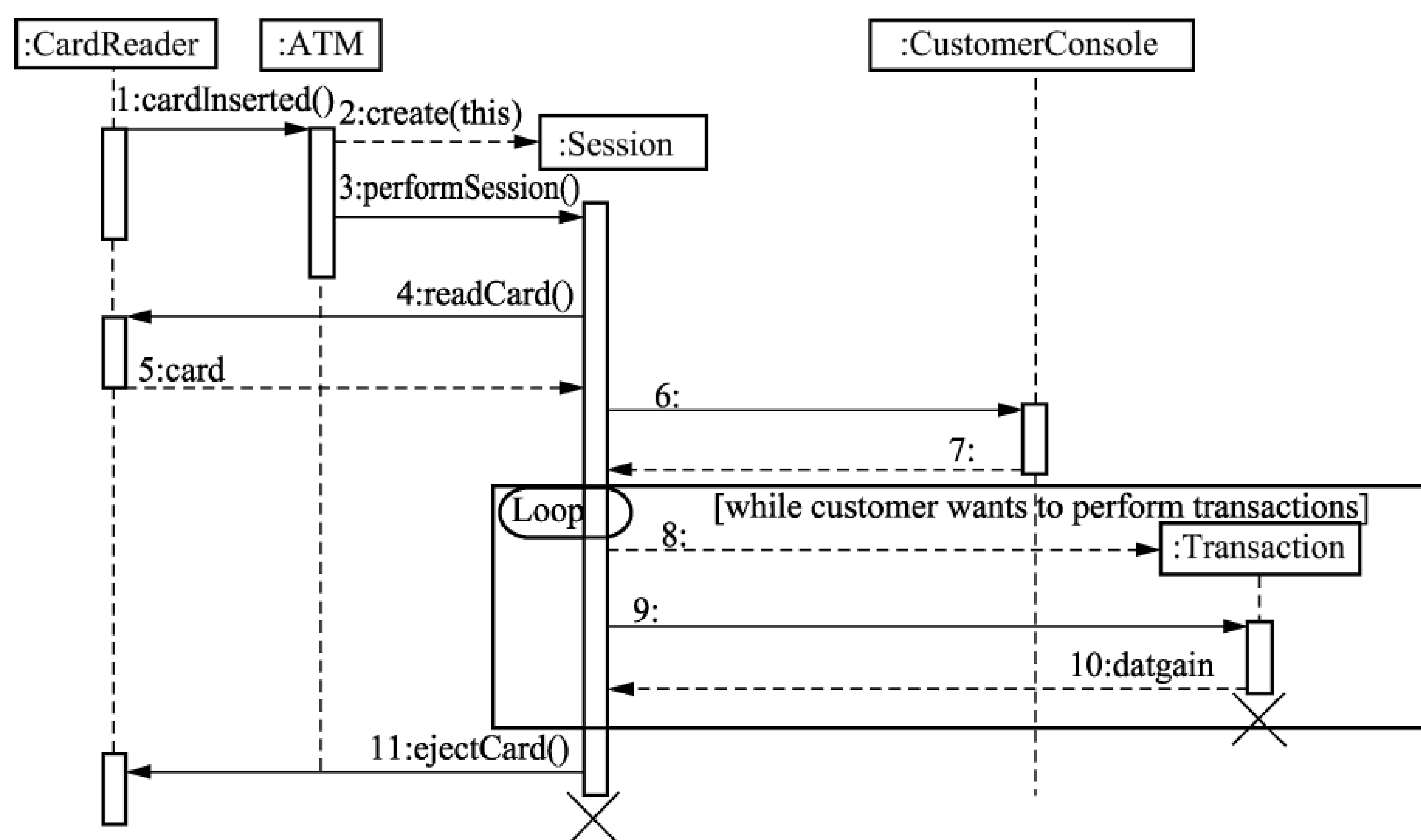


图 3-31　一次会话的序列图(无验证消息)

表 3-7　可能的消息名称列表

名　称	说　明	名　称	说　明
CardInserted()	ATM 卡已插入	performTransaction()	执行事务
performSession()	执行会话	readCard()	读卡
readPIN()	读取个人验证码	PIN	个人验证码信息
Create(atm, this, card, pin)	为当前会话创建事务	create(this)	为当前 ATM 创建会话
Card	ATM 卡信息	doAgain	执行下一个事务
ejectCard()	弹出 ATM 卡		

9. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2008 年 11 月试题三)

【说明】

在线会议审稿系统(Online Reviewing System，ORS)主要处理会议前期的投稿和审稿事务，其功能描述如下。

(1) 用户在初始使用系统时，必须在系统中注册(Register)成为作者或审稿人。

(2) 作者登录(Login)后提交稿件和浏览稿件审阅结果。提交稿件必须在规定提交时间范围内，其过程为输入标题和摘要、选择稿件所属主题类型、选择稿件所在位置(存储位置)。上述几步若未完成，则重复；若完成，则上传稿件至数据库中，系统发送通知。

(3) 审稿人登录后可设置兴趣领域、审阅稿件给出意见以及罗列录用和(或)拒绝的稿件。

(4) 会议委员会主席是一个特殊审稿人，可以浏览提交的稿件、给审稿人分配稿件、罗列录用和(或)拒绝的稿件及关闭审稿过程。其中关闭审稿过程需包括罗列录用和(或)拒绝的稿件。

系统采用面向对象方法开发，使用 UML 进行建模。在建模用例图时，常用的方式是先识别参与者，然后确定参与者如何使用系统来确定用例，每个用例可以构造一个活动图。参与者名称、用例名称和活动名称分别参见表 3-8~表 3-10。系统的部分用例图和提交稿件过程的活动图分别如图 3-32 和图 3-33 所示。

表 3-8　参与者名称

名　称	说　明	名　称	说　明
user	用户	author	作者
reviewer	审稿人	PCChair	委员会主席

表 3-9　用例名称

名　称	说　明	名　称	说　明
login	登录系统	register	注册
submit paper	提交稿件	browse review results	浏览稿件审阅结果
close reviewing process	关闭审稿过程	assign paper to reviewer	分配稿件给审稿人
set preferences	设定兴趣领域	enter review	审阅稿件并给出意见
list accepted/rejected papers	罗列录用或/和拒绝的稿件	browse submitted papers	浏览提交的稿件

表 3-10　活动名称

名　称	说　明	名　称	说　明
select paper location	选择稿件位置	upload paper	上传稿件
select subject group	选择主题类型	send notification	发送通知
enter title and abstract	输入标题和摘要		

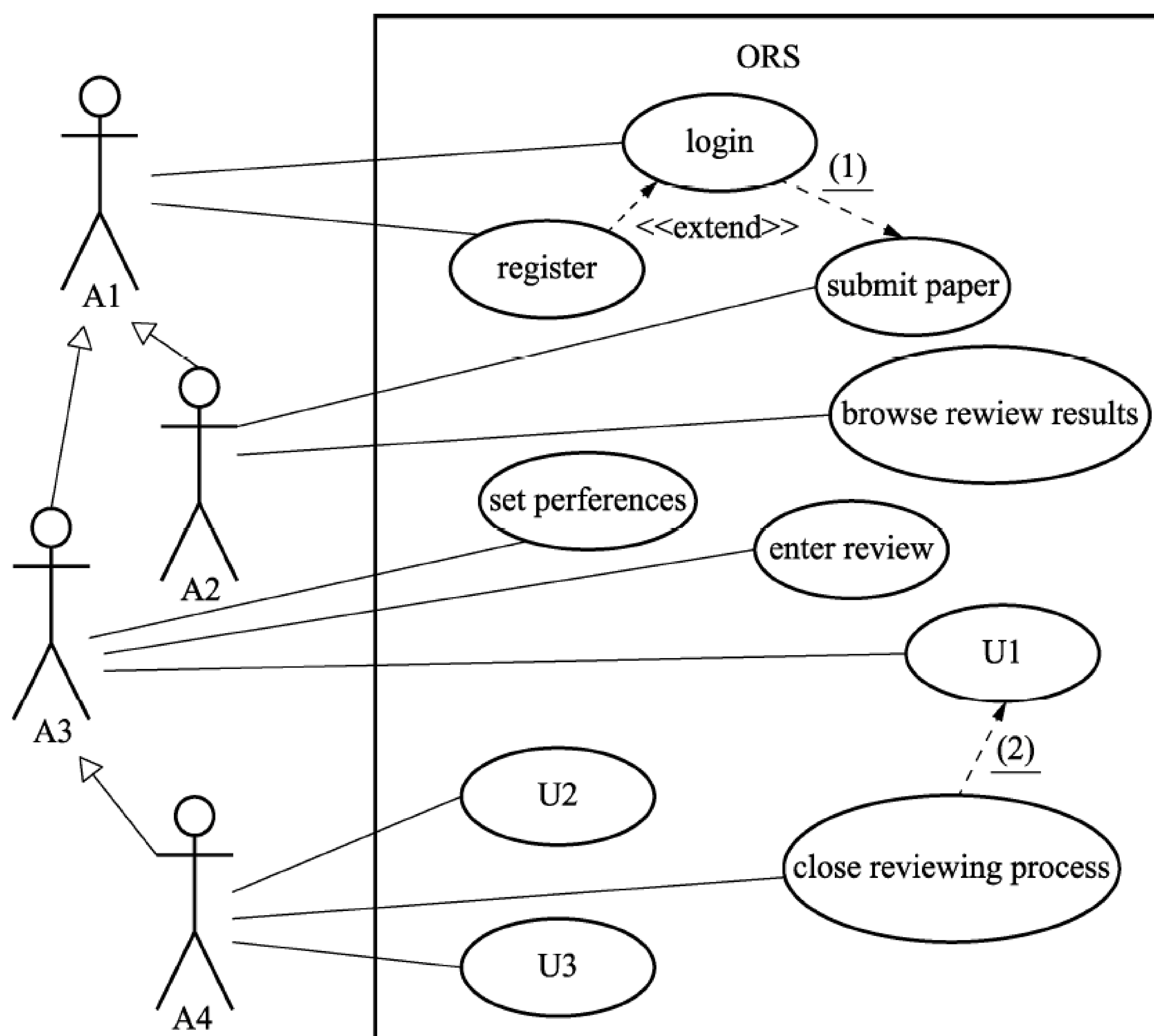


图 3-32　ORS 用例图

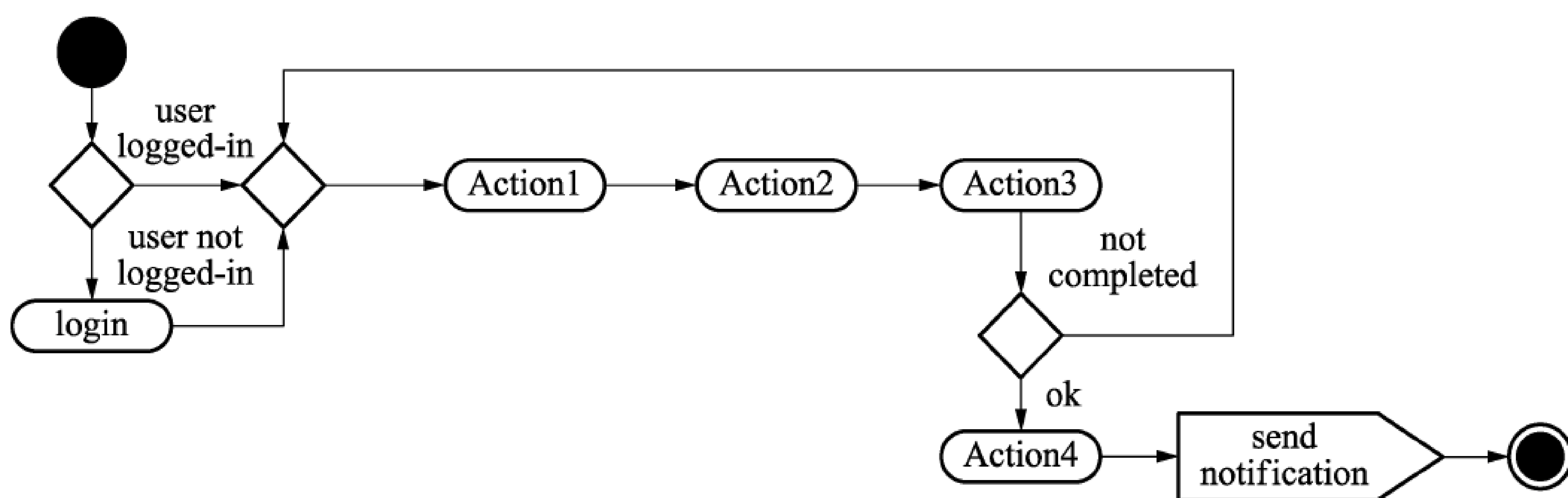


图 3-33　提交稿件过程的活动图

【问题 1】

根据说明中的描述，使用表 3-8 中的英文名称，给出图 3-32 中 A1~A4 所对应的参与者。

【问题 2】

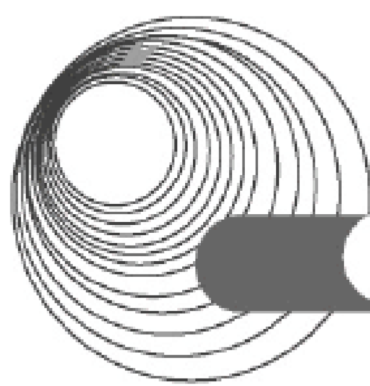
根据说明中的描述，使用表 3-9 中的英文名称，给出图 3-32 中 U1~U3 所对应的用例。

【问题 3】

根据说明中的描述，给出图 3-32 中(1)和(2)所对应的关系。

【问题 4】

根据说明中的描述，使用表 3-8 和表 3-9 中的英文名称，给出图 3-33 中 Action1~Action4 对应的活动。



10. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2008 年 5 月试题三)

【说明】

某汽车停车场欲建立一个信息系统，已经调查到的需求如下。

(1) 在停车场的入口和出口分别安装一个自动栏杆、一台停车卡打印机、一台读卡器和一个车辆通过传感器，示意图如图 3-34 所示。



图 3-34　停车场示意图

(2) 当汽车到达入口时，驾驶员按下停车卡打印机的按钮获取停车卡。当驾驶员拿走停车卡后，系统命令栏杆自动抬起；汽车通过入口后，入口处的传感器通知系统发出命令，栏杆自动放下。

(3) 在停车场内分布着若干个付款机器。驾驶员将在入口处获取的停车卡插入付款机器，并缴纳停车费。付清停车费之后将获得一张出场卡，用于离开停车场。

(4) 当汽车到达出口时，驾驶员将出场卡插入出口处的读卡器。如果这张卡是有效的，系统命令栏杆自动抬起；汽车通过出口后，出口传感器通知系统发出命令，栏杆自动放下。若这张卡是无效的，系统不发出栏杆抬起命令而发出警告信号。

(5) 系统自动记录停车场内空闲的停车位数量。若停车场当前没有车位，系统将在入口处显示“车位已满”信息。这时，停车卡打印机将不再出卡，只允许场内汽车出场。

根据上述描述，采用面向对象方法对其进行分析与设计，得到了表 3-11 所示的类/用例/状态列表、图 3-35 所示的用例图、图 3-36 所示的初始类图以及图 3-37 所示的描述入口自动栏杆行为的 UML 状态图。

表 3-11　类/用例/状态列表

用 例 名	说　明	类　名	说　明	状 态 名	说　明
Car entry	汽车进入停车场	Central Computer	停车场信息系统	Idle	空闲状态，汽车可以进入停车场
Car exit	汽车离开停车场	PaymentMachine	付款机器	Disable	没有车位
Report statistics	记录停车场的相关信息	CarPark	停车场，保存车位信息	Await Entry	等待汽车进入
		Barrier	自动护栏	Await Ticket Take	等待打印停车卡
Car entry when full	没有车位时，汽车请求进入停车场	EntryBarrier	入口的护栏	Await Enable	等待停车场内有空闲车位
		ExitBarrier	出口的护栏		

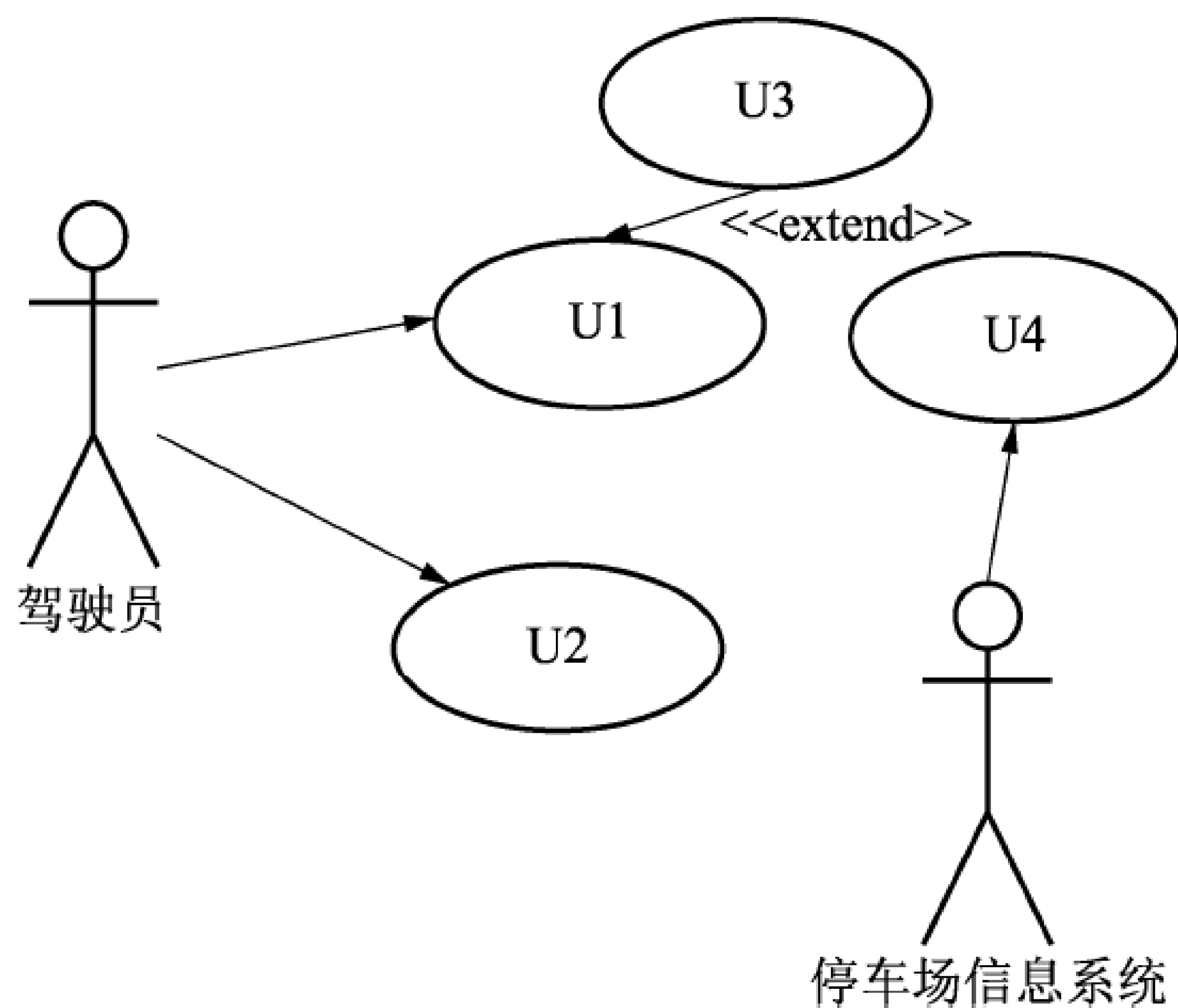


图 3-35　用例图

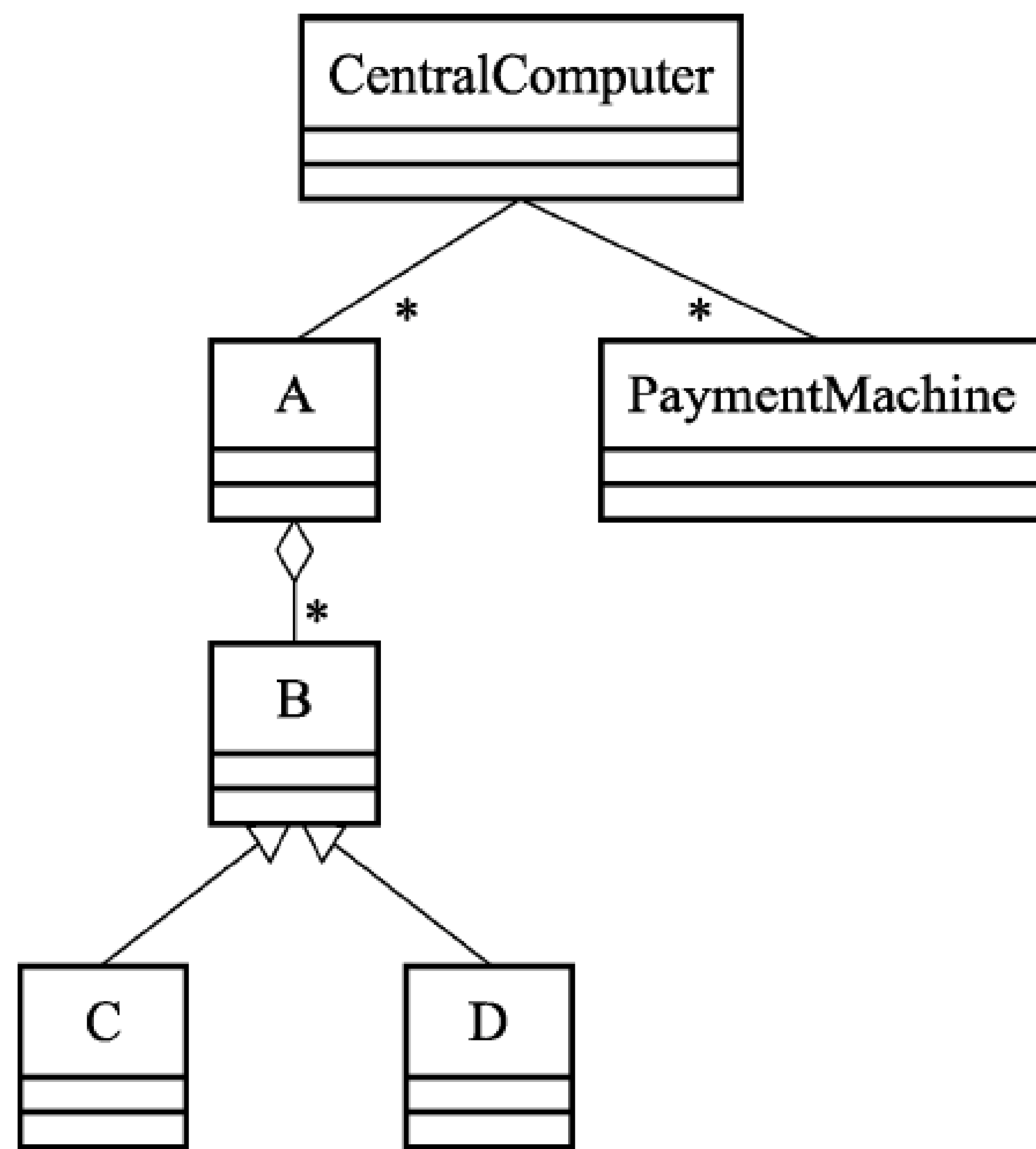


图 3-36　初始类图

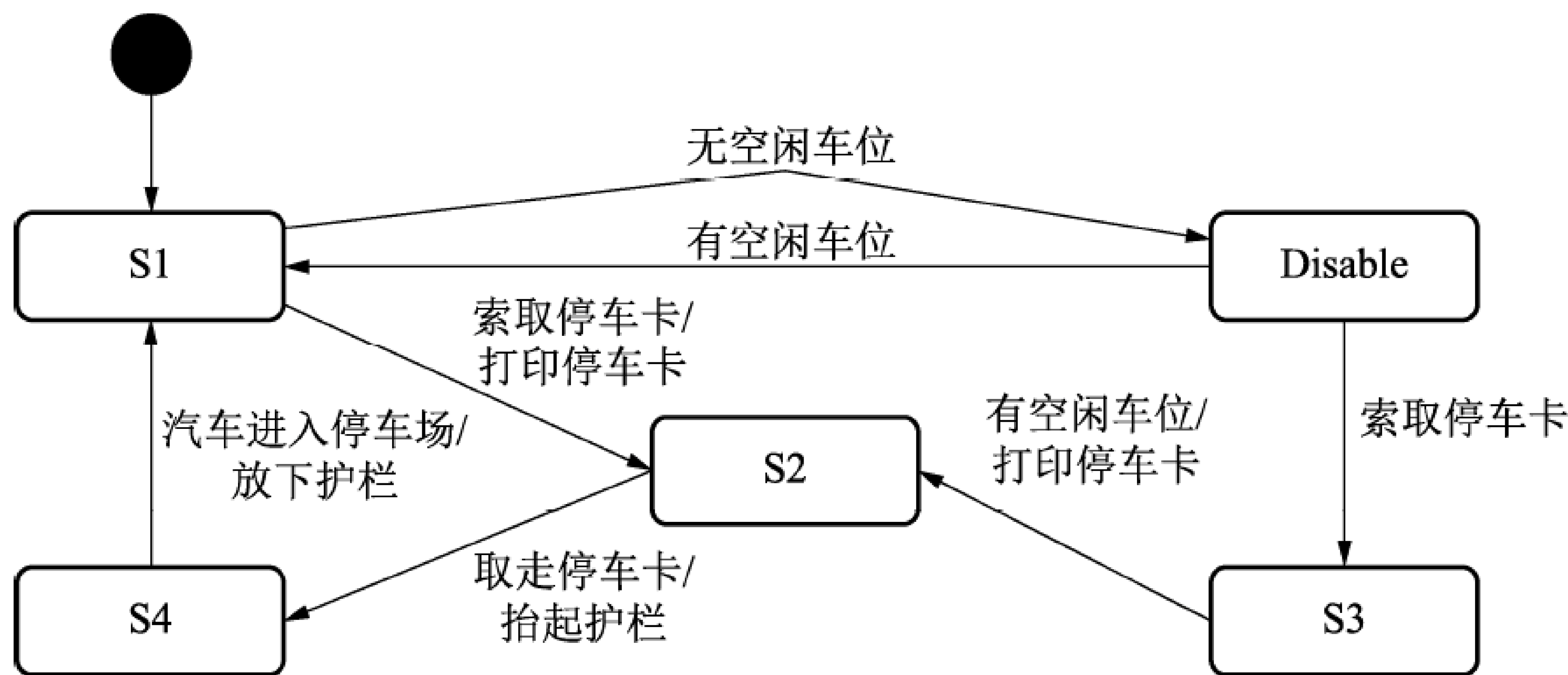


图 3-37　入口自动栏杆行为 UML 状态图

【问题 1】

根据说明中的描述，使用表 3-11 中的用例名，给出图 3-35 中 U1~U3 所对应的用例。

【问题 2】

根据说明中的描述，使用表 3-11 中的类名，给出图 3-36 中 A~D 所对应的类。

【问题 3】

根据说明中的描述，使用表 3-11 中的状态名，给出图 3-37 中 S1~S4 所对应的状态。

【问题 4】

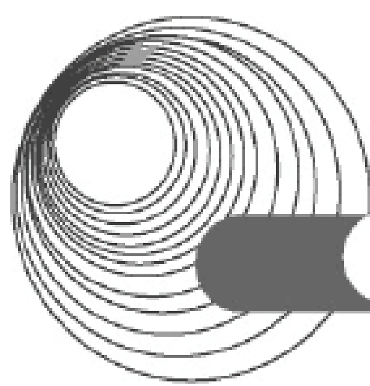
简要解释图 3-35 中用例 U1 和 U3 之间的 extend 关系的内涵。

11. 阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。(2007 年 11 月试题三)

【说明】

已知某唱片播放器不仅可以播放唱片，而且可以连接计算机并把计算机中的歌曲刻录到唱片上(同步歌曲)。连接计算机的过程中还可自动完成充电。

关于唱片，还有以下描述信息。



(1) 每首歌曲的描述信息包括歌曲的名字、谱写这首歌曲的艺术家及演奏这首歌曲的艺术家。只有两首歌曲的这 3 部分信息完全相同时，才认为它们是同一首歌曲。艺术家可能是一名歌手或一支由 2 名或 2 名以上的歌手所组成的乐队。一名歌手可以不属于任何乐队，也可以属于一个或多个乐队。

(2) 每张唱片由多条音轨构成。一条音轨中只包含一首歌曲或为空，一首歌曲可分布在多条音轨上；同一首歌曲在一张唱片中最多只能出现一次。

(3) 每条音轨都有一个开始位置和持续时间。每张唱片上的音轨次序是非常重要的，因此对于任意一条音轨，播放器需要准确地知道它的下一条音轨和上一条音轨是什么(如果存在的话)。

根据上述描述，采用面向对象方法对其进行分析与设计，得到如表 3-12 所示的类列表、如图 3-38 所示的初始类图以及如图 3-39 所示的描述播放器行为的 UML 状态图。

表 3-12　类列表

类　名	说　明
Artist	艺术家
Song	歌曲
Band	乐队
Musician	歌手
Track	音轨
Album	唱片

【问题 1】

根据说明中的描述，使用表 3-12 中的类名，给出图 3-38 中 A~F 所对应的类。

【问题 2】

根据说明中的描述，给出图 3-38 中(1)~(6)处的多重度。

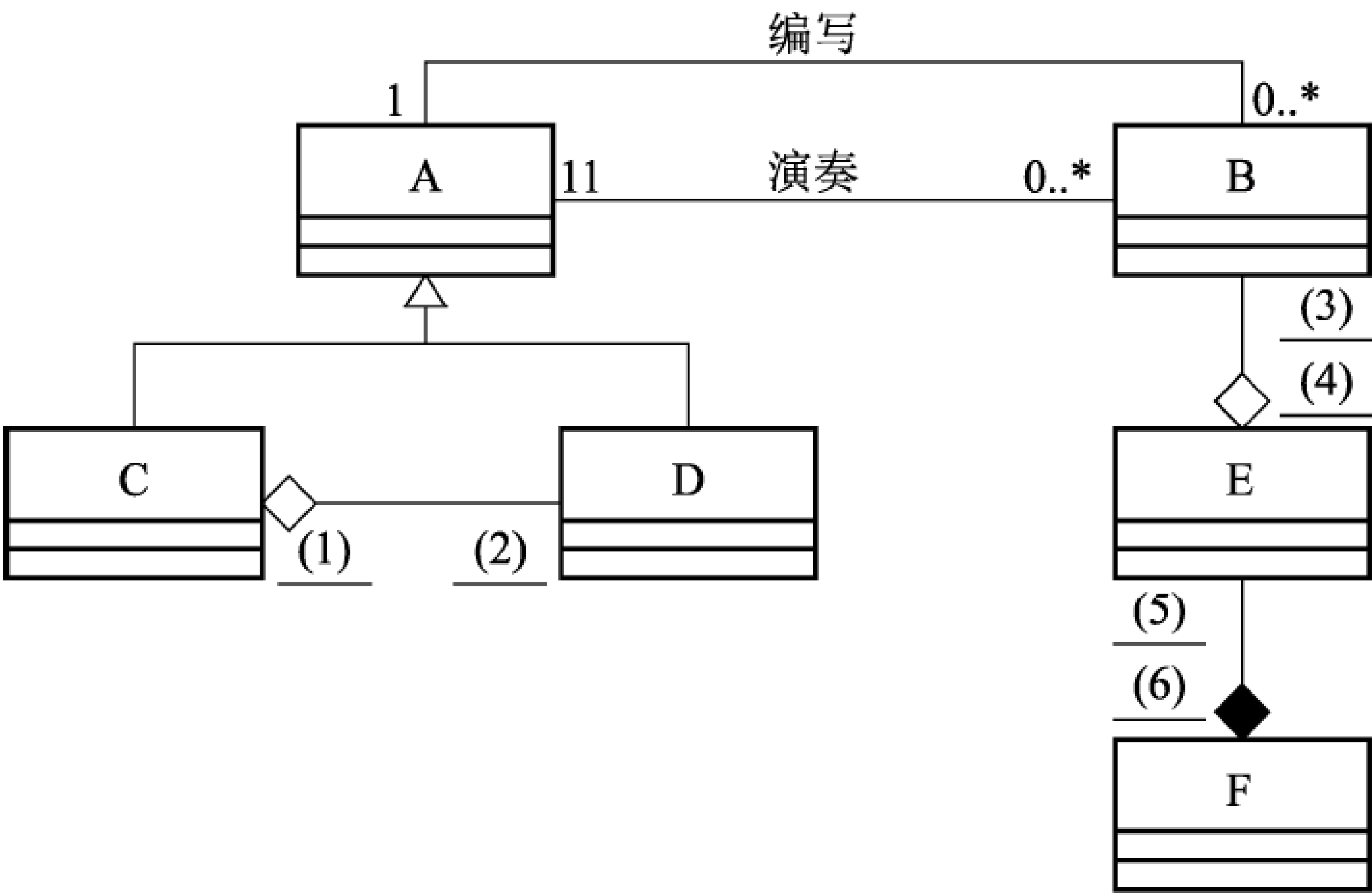


图 3-38　初始类图

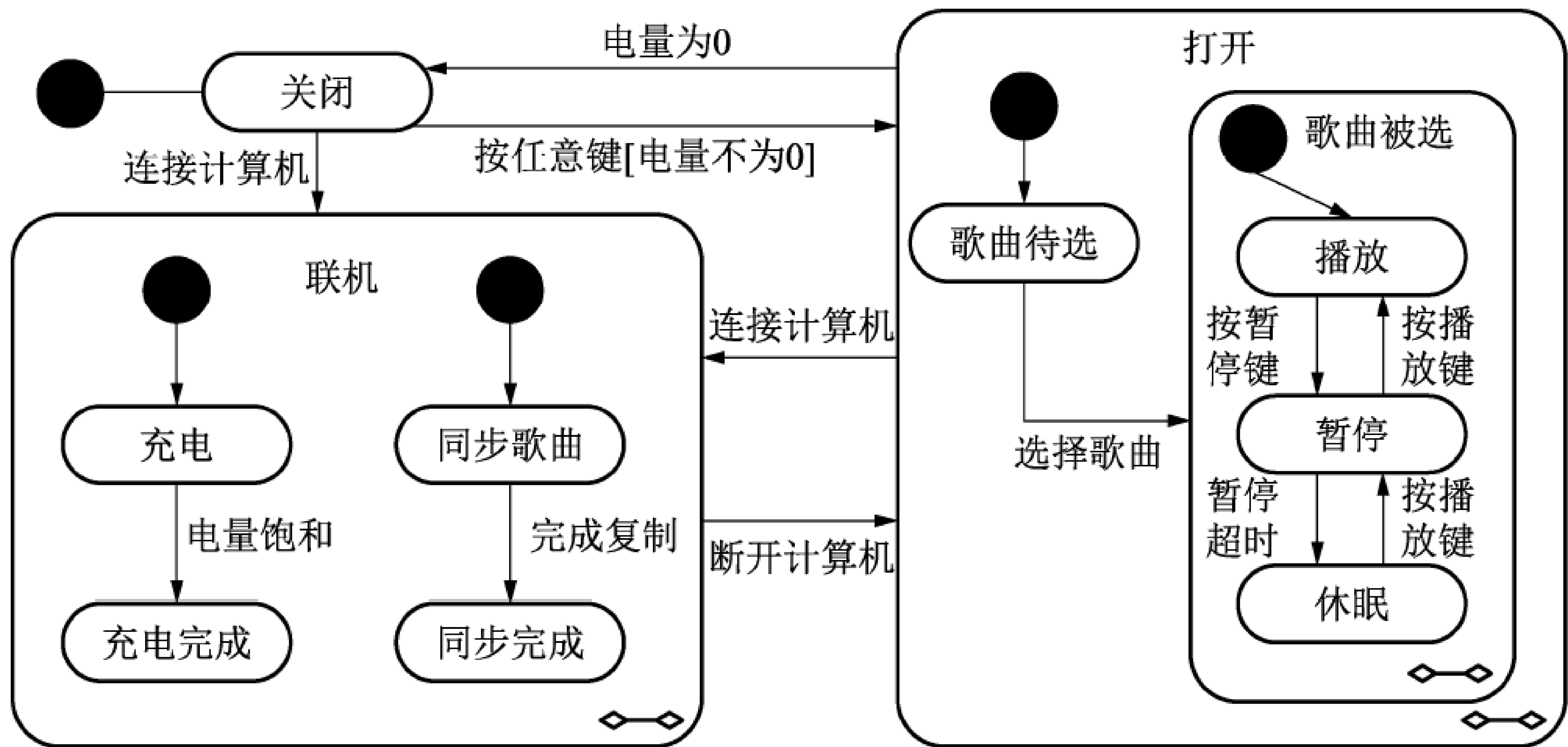


图 3-39　播放器行为 UML 状态图

【问题 3】

图 3-38 中缺少了一条关联，请指出这条关联两端所对应的类以及每一端的多重度，填入表 3-13 中。

表 3-13　题 11 问题 3 表

类	多 重 度

【问题 4】

根据图 3-39 所示的播放器行为 UML 状态图，给出从“关闭”状态到“播放”状态所经过的最短事件序列(假设电池一开始就是有电的)。

12. 阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2007 年 5 月试题三)

【说明】

某图书管理系统的主要功能如下。

(1) 图书管理系统的资源目录中记录着所有可供读者借阅的资源，每项资源都有一个唯一的索引号。系统需登记每项资源的名称、出版时间和资源状态(可借阅或已借出)。

(2) 资源可以分为两类：图书和唱片。对于图书，系统需登记作者和页数；对于唱片，系统需登记演唱者和介质类型(CD 或者磁带)。

(3) 读者信息保存在图书管理系统的读者信息数据库中，记录的信息包括读者的识别码和读者姓名。系统为每个读者创建了一个借书记录文件，用来保存读者所借资源的相关信息。

现采用面向对象方法开发该图书管理系统。识别类是面向对象分析的第一步。比较常用的识别类的方法是寻找问题描述中的名词，再根据相关规则从这些名词中删除不可能成为类的名词，最终得到构成该系统的类。表 3-14 给出了说明中出现的所有名词。

表 3-14　图书管理系统

图书管理系统	资源目录	读　者	资　源
索引号	系统	名称	出版时间
资源状态	图书	唱片	作者
页数	演唱者	介质类型	CD
磁带	读者信息	读者信息数据库	识别码
姓名	借书记录文件	信息	

通过对表 3-14 中的名词进行分析，最终得到如图 3-40 所示的 UML 类图。类的说明如表 3-15 所示。

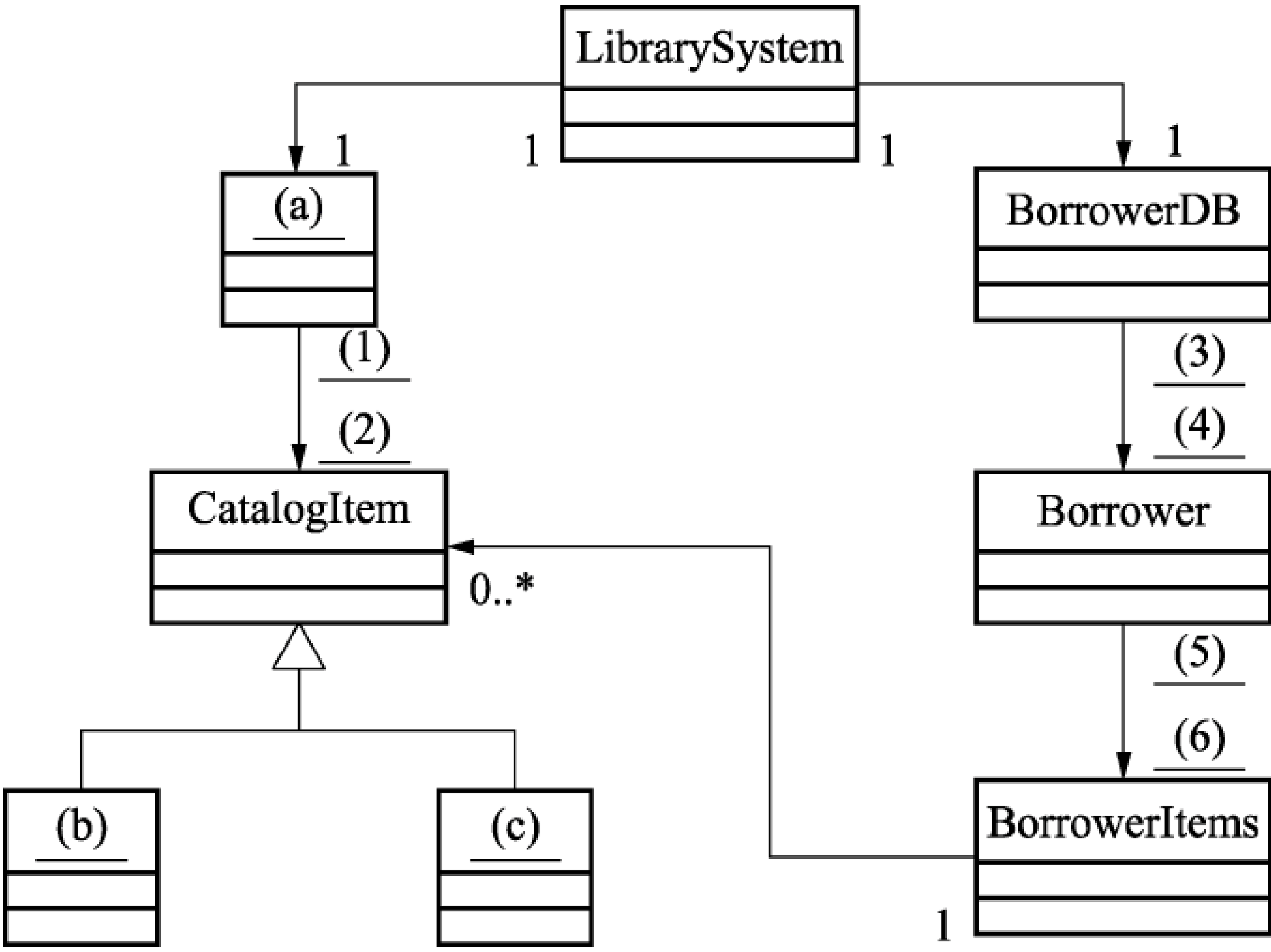


图 3-40　UML 类图

【问题 1】

表 3-15 所给出的类并不完整，根据说明和表 3-14，将图 3-40 中的(a)~(c)处补充完整。

表 3-15　类的说明

类　名	说　明
LibrarySystem	图书管理系统
BorrowerDB	保存读者信息的数据库
CatalogItem	资源目录中保存的每项资源
Borrower	读者
BorrowerItems	为每个读者创建的借书记录文件

【问题 2】

根据说明中的描述，给出图 3-40 所示的类 CatalogItem 以及(b)、(c)处所对应的类的关键属性(使用表 3-14 中给出的词汇)。其中，CatalogItem 有 4 个关键属性；(b)、(c)处对应的类各有 2 个关键属性。

【问题 3】

识别关联的多重度是面向对象建模过程中的一个重要步骤。根据说明中给出的描述，完成图 3-40 中的(1)~(6)。

13. 阅读以下说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2006 年 11 月试题三)

【说明】

S 公司开办了在线电子商务网站，主要为各注册的商家提供在线商品销售功能。为更好地吸引用户，S 公司计划为注册的商家提供商品(Commodity)促销(Promotion)功能。商品的分类(Category)不同，促销的方式和内容也会有所不同。

注册商家可发布促销信息。商家首先要在自己所销售的商品的分类中，选择促销涉及的某一具体分类，然后选出该分类的一个或多个商品(一种商品仅属于一种分类)，接着制定出一个比较优惠的折扣政策和促销活动的优惠时间，最后由系统生成促销信息，并将该促销信息公布在网站上。

商家发布促销信息后，网站的注册用户便可通过网站购买促销商品。用户可选择参与某一个促销活动，并选择具体的促销商品，输入购买数量等购买信息。系统生成相应的一份促销订单(POrder)。只要用户在优惠活动的时间范围内，通过网站提供的在线支付系统，确认在线支付该促销订单(即完成支付)，就可以优惠的价格完成商品的购买活动，否则该促销订单失效。

系统采用面向对象方法开发，系统中的类及类之间的关系用 UML 类图表示，图 3-41 是该系统类图中的一部分；系统的动态行为采用 UML 序列图表示，图 3-42 是发布促销的序列图。

【问题 1】

识别关联的多重度是面向对象建模过程中的一个重要步骤。根据说明中给出的描述，完成图 3-41 中的(1)~(6)。

【问题 2】

请从表 3-16 中选择方法，完成图 3-42 中的(7)~(10)。

【问题 3】

关联(Association)和聚集(Aggregation)是 UML 中两种非常重要的关系。请说明关联和聚集的关系，并说明其不同点。

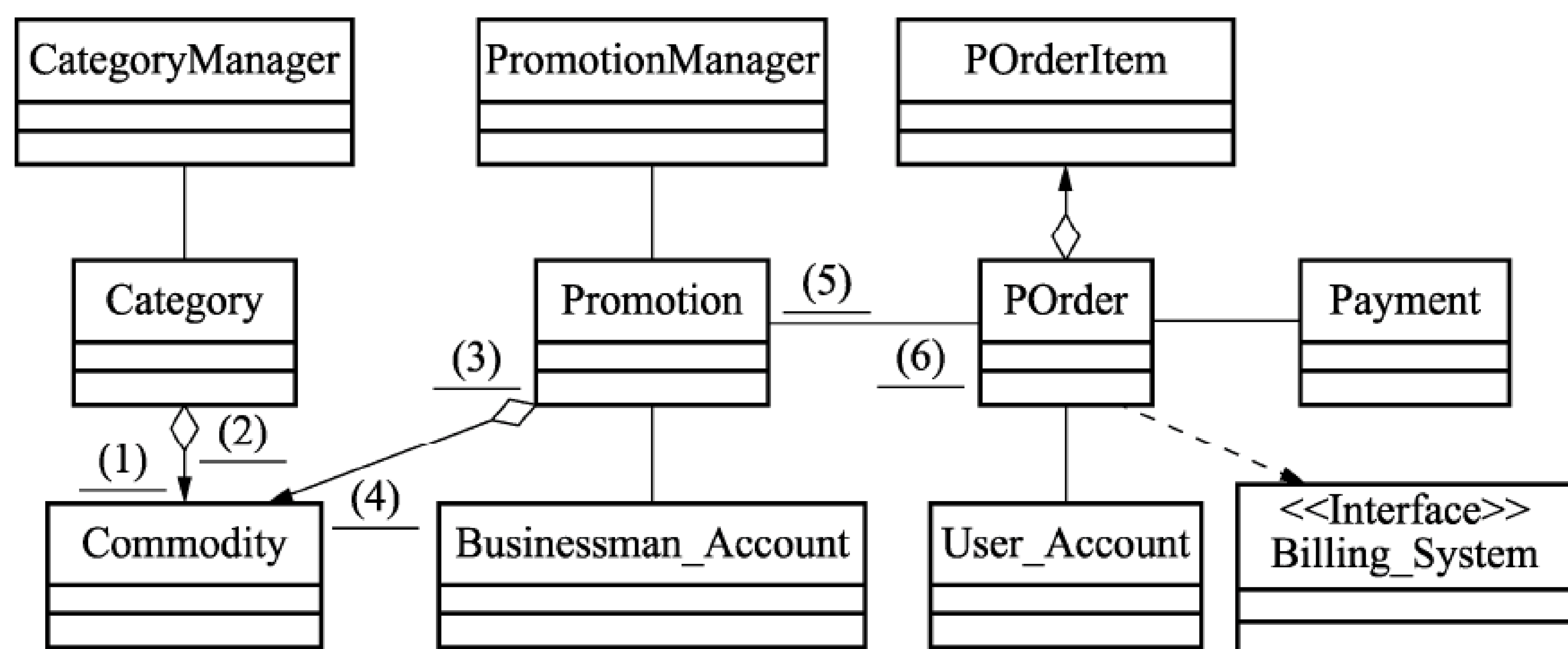


图 3-41　在线促销系统部分类图

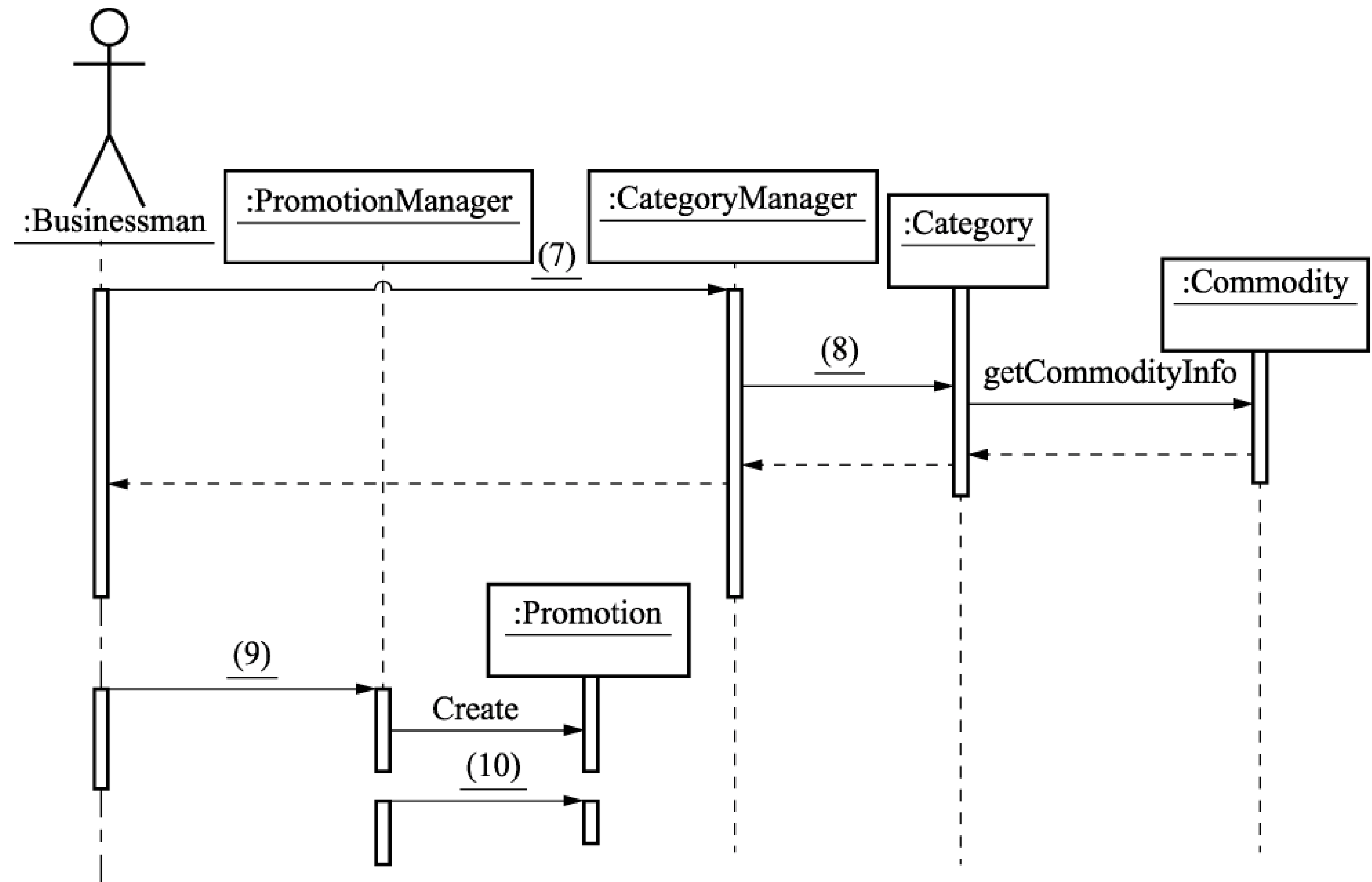


图 3-42　发布促销序列图

表 3-16　可选消息列表

功能描述	方 法 名
向促销订单中添加所选的商品	buyCommodities
向促销订单中添加要促销的商品	addCommodities
查找某个促销的所有促销订单信息列表	getPromotionOrders
生成商品信息	createCommodity
查找某个分类中某商家的所有商品信息列表	getCommodities
生成促销信息	createPromotion
生成促销订单信息	createPOrder
查找某个分类的所有促销信息列表	getCategoryPromotion
查找某商家所销售的所有分类列表	getCategories
查找某个促销所涉及的所有商品信息列表	getPromotionCommodities

14. 阅读下列说明以及 UML 类图，回答问题，将解答填入答题纸的对应栏内。(2006 年 5 月试题二)

【说明】

某客户信息管理系统中保存着以下两类客户的信息。

(1) 个人客户。对于这类客户，系统保存了其客户标识(由系统生成)和基本信息(包括姓名、住宅电话和 E-mail)。

(2) 集团客户。集团客户可以创建和管理自己的若干名联系人。对于这类客户，系统除了保存其客户标识(由系统生成)之外，也保存了其联系人的信息。联系人的信息包括姓名、住宅电话、E-mail、办公电话及职位。

该系统除了可以保存客户信息之外，还具有以下功能。

(1) 向系统中添加客户(addCustomer)。

(2) 根据给定的客户标识，在系统中查找该客户(getCustomer)。

(3) 根据给定的客户标识，从系统中删除该客户(removeCustomer)。

(4) 创建新的联系人(addContact)。

(5) 在系统中查找指定的联系人(getContact)。

(6) 从系统中删除指定的联系人(removeContact)。

该系统采用面向对象方法进行开发。在面向对象分析阶段，根据上述描述，得到如表 3-17 所示的类及说明。

表 3-17　类及说明

类　名	说　明
CustomerInformationSystem	客户信息管理系统
IndividualCustomer	个人客户
InstitutionalCustomer	集团客户
Contact	联系人

描述该客户信息管理系统的 UML 类图如图 3-43 所示。

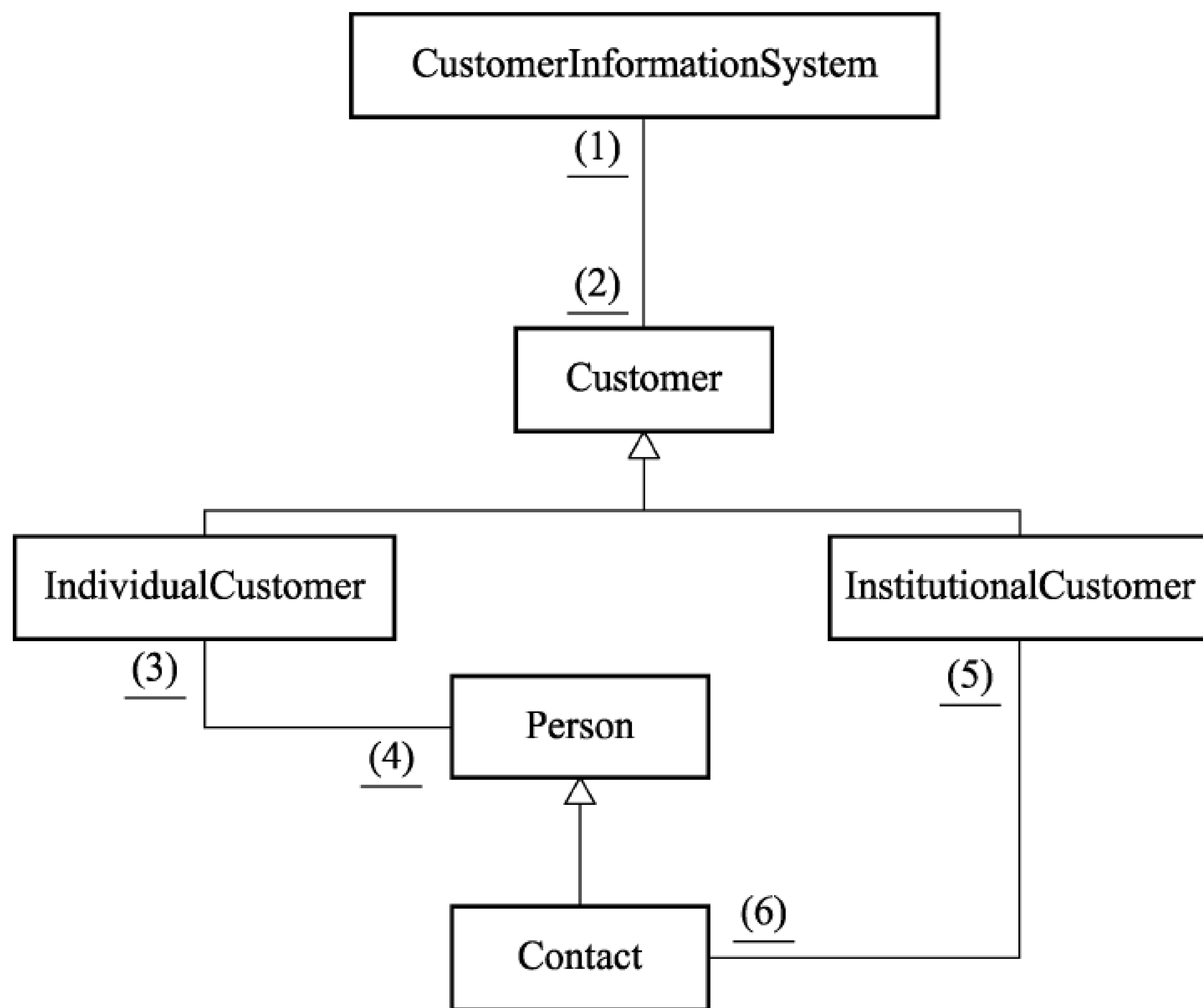


图 3-43　UML 类图

【问题 1】

请使用说明中的描述，给出图 3-43 中 Customer 类和 Person 类的属性。

【问题 2】

识别关联的多重度是面向对象建模过程中的一个重要步骤，请根据说明中给出的描述，完成图 3-43 中的(1)~(6)。

【问题 3】

根据说明中的叙述，抽象出如表 3-18 所示的方法，请指出图 3-43 中的 Customer-InformationSystem 类和 InstitutionalCustomer 类应分别具有其中的哪些方法。

表 3-18 方法及其描述

功能描述	方 法 名
向系统中添加客户	addCustomer
根据给定的客户标识，在系统中查找该客户	getCustomer
根据给定的客户标识，从系统中删除该客户	removeCustomer
创建新的联系人	addContact
在系统中查找指定的联系人	getContact
从系统中删除指定的联系人	removeContact

3.1.4 同步练习参考答案

1.

【问题 1】

UC1：CheckAvailability。 UC2：MakeReservation。 UC3：GetDiscount。

UC4：ManageCashPayment。 UC5：ManageCrCardPayment。 UC6：CalculateRefund。

【问题 2】

C1：NationalPark。 C2：Rate。 C3：TicketingOfficer。

C4：Payment。 C5：Discount。 C6：CashPayment。

C7：CreditCardPayment。

【问题 3】

修改方案 1：增加一个新类，该类与 ReservationItem 类之间有关联关系。

修改方案 2：修改 Rate 类，使其具有计算赔偿金的功能。

2.

【问题 1】

A1：顾客。 A2：订单处理人员。 A3：派送人员。 U1：收货。 U2：派单。

【问题 2】

C1：Customer。 C2：Order。 C3：Product。

(1)：1。 (2)：0..n 或 0..*。 (3)：0..n 或 0..*。 (4)：1..n 或 1..*。

【问题 3】

C2 的属性：volume、delivery date、form of payment。

C3 的属性：cubic volume、cost price、sale price。

3.

【问题 1】

U1：使用常规卡行驶。 U2：使用单次卡行驶。 (1)：<<extend>>

【问题 2】

C1：RoadSegment。　C2：Trajectory。　C3：Card。

C4：RegularCard。　C5：PrepaidCard。　C6：MinitripCard。

(2)：1。　(3)：1..3。

【问题 3】

RoadSegment 的属性：Distance。

Trajectory 的属性：Entry、Exit、DateOfEntry。

Card 的属性：UnitPrice、ValidPeriod。

4.

【问题 1】

U1：移动元素。　U2：调整元素大小。

注：U1 和 U2 的答案可以互换。

(1)：<<extend>>。　(2)：<<extend>>。

【问题 2】

C1：创建工具。　C2：选择工具。　C3：线条工具。　C4：矩形工具。

C5：椭圆工具。　C6：线条。　C7：矩形。　C8：椭圆。

注：C3~C5 的答案可以互换；C6~C8 的答案可以互换。

(3)：0..1。　(4)：1。　(5)：1。　(6)：1..*或*。

【问题 3】

桥接模式将抽象部分和它的实现部分分离，使它们都可以独立地变化，对一个抽象的实现部分的修改应该对使用它的程序不产生影响。

5.

【问题 1】

C1：付款方式。　C2：处方。　C3：信用卡。　C4：支付宝账户。　C5：药品。

(1)：1。　(2)：0..*。　(3)：1。　(4)：1..*。　(5)：0..*。　(6)：1。

【问题 2】

S1：审核中。　S2：无法审核。　S3：医生信息无效。　S4：处方无效。

(7)：医生信息不正确。　(8)：医生信息正确。

(9)：医生回复处方无效。　(9)：医生没有在 7 天内给出确认答复。

【问题 3】

◆：表示组合，◇：表示聚集。

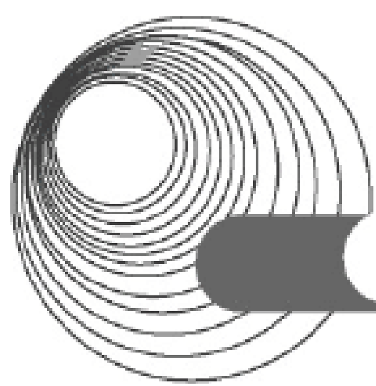
两者之间的区别为：在组合关系中，整体对象与部分对象具有同一的生存周期，当整体对象不存在时，部分对象也不存在；而在聚集关系中，对整体对象和部分对象没有这样的要求。

6.

【问题 1】

A1：乘客。　A2：服务技术人员。　U1：支付。

(1)：<<include>>。　(2)：<<include>>。



【问题 2】

C1：键盘。　　C2：目的地键盘。　　C3：车票键盘。　　C4：继续/取消键盘。

(3)~(6)：1。

【问题 3】

使用 Mediator 模式，可以使各个对象间的耦合松散，只需关心和 Mediator 的关系，使多对多的关系变成一对多的关系，降低系统的复杂性，提高可修改扩展性。

7.

【问题 1】

A1：工资系统。　　A2：菜单管理员。

【问题 2】

(1) 查看今日特价，参与者：员工。

(2) 注册工资支付，参与者：顾客和工资系统。

(3) 生成付费请求，参与者：餐厅员工和工资系统。

(4) 管理菜单，参与者：菜单管理员。

【问题 3】

(1)：◇。　　(2)：登录。　　(3)：▬ 或 ▮。　　(4)：▬ 或 ▮。

【问题 4】

泛化关系。泛化是一种一般—特殊关系，特殊元素(子元素)的对象可替代一般元素(父元素)的对象。本题中顾客和员工就是一般—特殊关系，且说明中有描述“系统的顾客是注册到系统的员工”。

8.

【问题 1】

A1：Customer。　　A2：Bank。

U1：Session。　　U2：Invalid PIN Process。　　U3：Transaction。

(1)：<<extend>>。

【问题 2】

6：readPIN()。　7：PIN。　8：Create(atm,this,card,pin)。　9：performTransaction()。

【问题 3】

它们之间是泛化关系。Transaction 是一个抽象泛化用例，具有其他事务类型共有的属性和行为，每个具体的事务类型继承它，并实现适合自己的特定操作。

9.

【问题 1】

A1：user。　A2：author。　A3：reviewer。　A4：PCChair。

【问题 2】

U1：list accepted/rejected papers。

U2：browse submitted papers。

U3：assign paper to reviewer。

【问题 3】

(1)：<<extend>>。　(2)：<<include>>。

【问题 4】

Action1：enter title and abstract。

Action2：select subject group。

Action3：select paper location。

Action4：upload paper。

10.

【问题 1】

U1：Car entry。　U2：Car exit。　U3：Car entry when full。

【问题 2】

A：CarPark。　B：Barrier。　C：EntryBarrier。　D：ExitBarrier。

【问题 3】

S1：Idle。　S2：Await Ticket Take。　S3：Await Enable。　S4：Await Entry。

【问题 4】

U3 是 U1 的扩展，当汽车要进入时判断是否有空车位。U1 和 U3 的 extend 关系表示一种聚集关系，具体为“组合”关系。它表示 U3 是 U1 状态的一个子集，U3 是 U1 的一部分。

11.

【问题 1】

A：Artist。　B：Song。　C：Band。　D：Musician。　E：Track。　F：Album。

【问题 2】

(1)：0..*。　(2)：2..*。　(3)：0..1。　(4)：1..*。　(5)：1..*。　(6)：1。

【问题 3】

类：Track。　多重度：0..1。

类：Track。　多重度：0..1。

【问题 4】

按任意键，选择歌曲。

12.

【问题 1】

(a)：资源目录。　(b)：图书。　(c)：唱片。

【问题 2】

CatalogItem 的属性：索引号、名称、出版时间、资源状态。

图书的属性：作者、页数。

唱片的属性：演唱者、介质类型。

【问题 3】

(1)：1。　(2)：0..*。　(3)：1。

(4)：0..*。　(5)：1。　(6)：1 或者 0..1。

13.

【问题 1】

(1)：0..*。　(2)：1。　(3)：0..*。

(4)：1..*。　　(5)：1。　　(6)：0..*。

【问题 2】

(7) :getCategories。　　(8) :getCommodities。

(9) :creatPromotion。　　(10) :addCommodities。

【问题 3】

关系：聚集是关联的特例。

不同点：聚集表示部分与整体关系的关联；若从生命周期的角度考虑，则关联对象的生命周期一般无必然关系，聚集的整体对象往往对部分对象的生命周期负责。

14.

【问题 1】

Customer 类的属性：客户标识。

Person 类的属性：姓名、住宅电话、E-mail。

【问题 2】

(1)：1。　　(2)：0..*。　　(3)：1。　　(4)：1。　　(5)：1。　　(6)：1..*。

【问题 3】

CustomerInformationSystem 类的方法：getCustomer、addCustomer、removeCustomer。

InstitutionalCustomer 类的方法：addContact、getContact、removeContact。

3.2 本 章 小 结

本章知识点在 2009 年的新大纲中改动不大。

根据近几年软件设计师水平考试试题分布情况来看，UML 分析与设计已经成为下午部分的必考题，占的分数也是一定的，几个小问题，一共 15 分。

UML 分析与设计的题目主要考查 UML 的用例图、类图、序列图及状态转换图，尤其是类图、类的属性和方法以及类间的各种关系等，还有一些图形符号，需要重点掌握。答题时要注意看清楚题目中给的几个图。

第 4 章　程序流程图

大纲要求：

要求掌握程序算法的描述方法。其中程序流程图是比较常用的方法，虽然近几年考查较少，不过建议掌握。

4.1　程序流程图的基本知识

4.1.1　考点辅导

程序流程图又称程序框图，是历史悠久、使用广泛的描述软件设计的方法，其主要优点是对控制流程的描绘很直观。

程序流程图近几年考查较少，不过还是需要掌握。解题时要认真阅读题中的说明，了解程序流程图所描述的功能及处理流程。

程序流程图中使用的主要结构包括顺序结构、选择结构(又称分支结构)和循环结构，如图 4-1 所示。值得注意的是，程序流程图中箭头代表的是控制流而不是数据流。

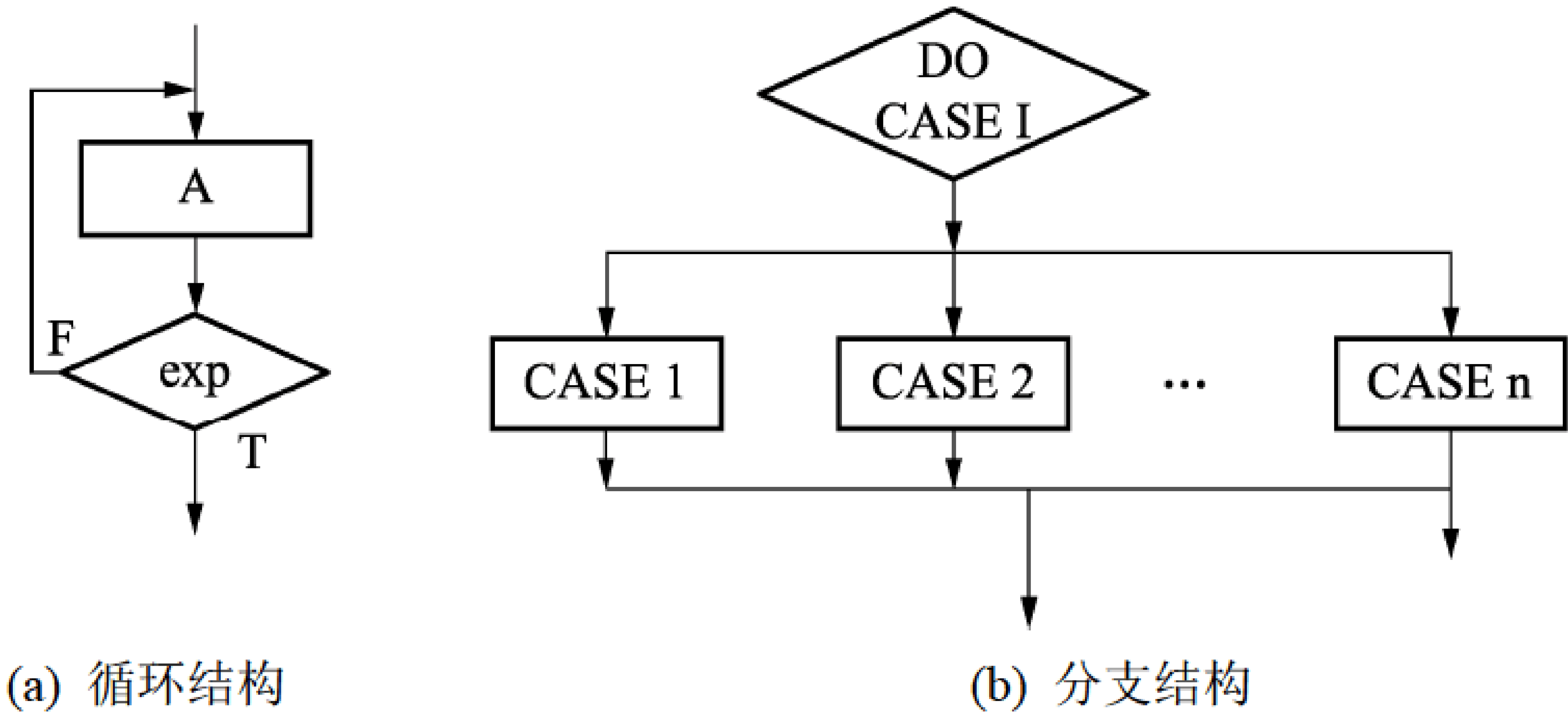


图 4-1　程序流程图

程序流程图中常用的符号如图 4-2 所示。

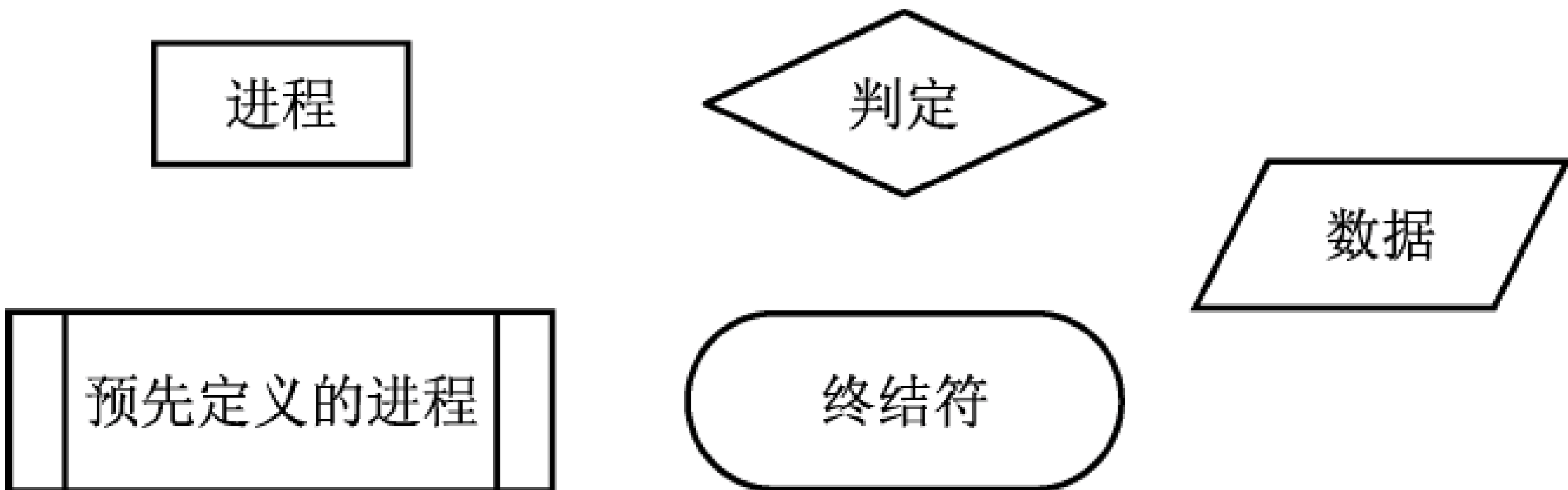
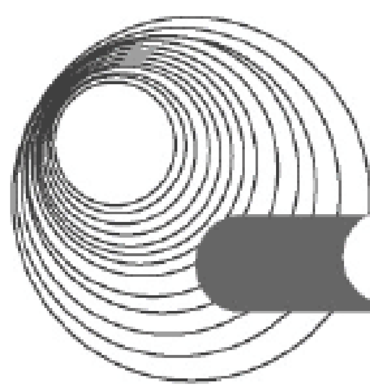


图 4-2　程序流程图中常用的符号



程序流程图的主要缺点如下。

(1) 程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

(2) 程序流程图中用箭头代表控制流，因此程序员不受约束，可以完全不顾结构程序设计的精神随意转移控制。

(3) 程序流程图不易表示数据结构。

4.1.2 典型例题分析

阅读下列说明和图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。

【说明】

某会议中心提供举办会议的场地设施和各种设备，供公司与各类组织机构租用。场地包括一个大型报告厅、一个小型报告厅以及诸多会议室。这些报告厅和会议室可提供的设备有投影仪、白板、视频播放/回放设备、计算机等。为了加强管理，该中心欲开发一会议预订系统，系统的主要功能如下。

(1) 检查可用性。客户提交预订请求后，检查预订表，判定所申请的场地是否在申请日期内可用；如果不可用，返回不可用信息。

(2) 临时预订。会议中心管理员收到客户预订请求的通知之后，提交确认。系统生成新临时预订存入预订表，并对新客户创建一条客户信息记录加以保存。根据客户记录给客户发送临时预订确认信息和支付定金要求。

(3) 分配设施与设备。根据临时预订或变更预订的设备和设施需求，分配所需设备(均能满足用户要求)和设施，更新相应的表和预订表。

(4) 确认预订。管理员收到客户支付定金的通知后，检查确认，更新预订表，根据客户记录给客户发送预订确认信息。

(5) 变更预订。客户还可以在支付余款前提交变更预订请求，对变更的预订请求检查可用性，如果可用，分配设施和设备；如果不可用，返回不可用信息。管理员确认变更后，根据客户记录给客户发送确认信息。

(6) 要求付款。管理员从预订表中查询距预订的会议时间两周内的预订，根据客户记录给满足条件的客户发送支付余款要求。

(7) 支付余款。管理员收到客户余款支付的通知后，检查确认，更新预订表中的已支付余款信息。

现采用结构化方法对会议预订系统进行分析与设计，获得如图 4-3 所示的上下文数据流图和图 4-4 所示的 0 层数据流图(不完整)。

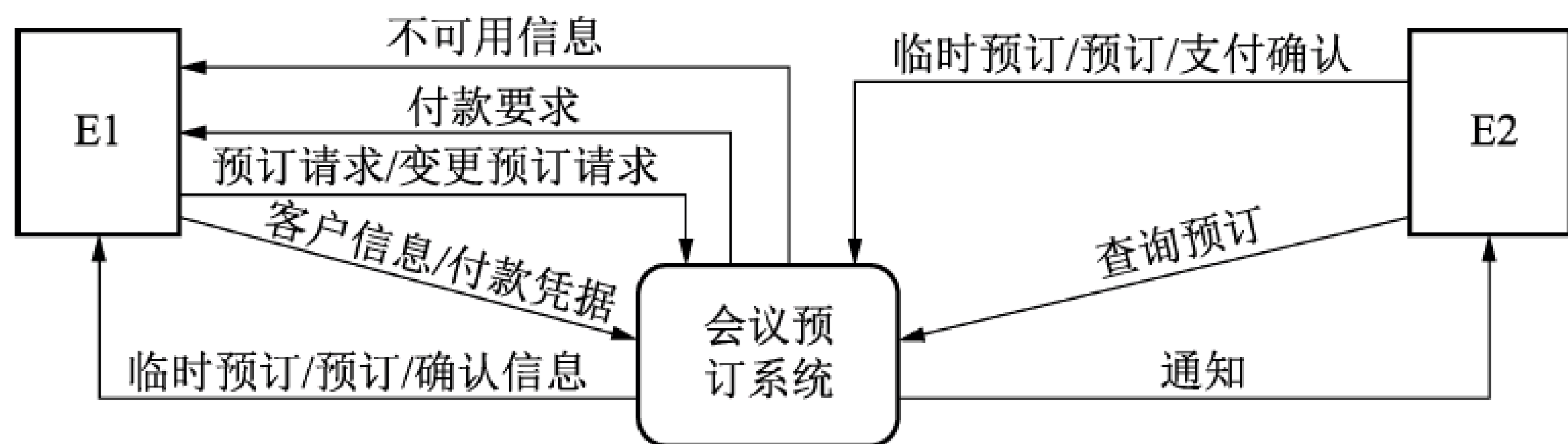


图 4-3　上下文数据流图

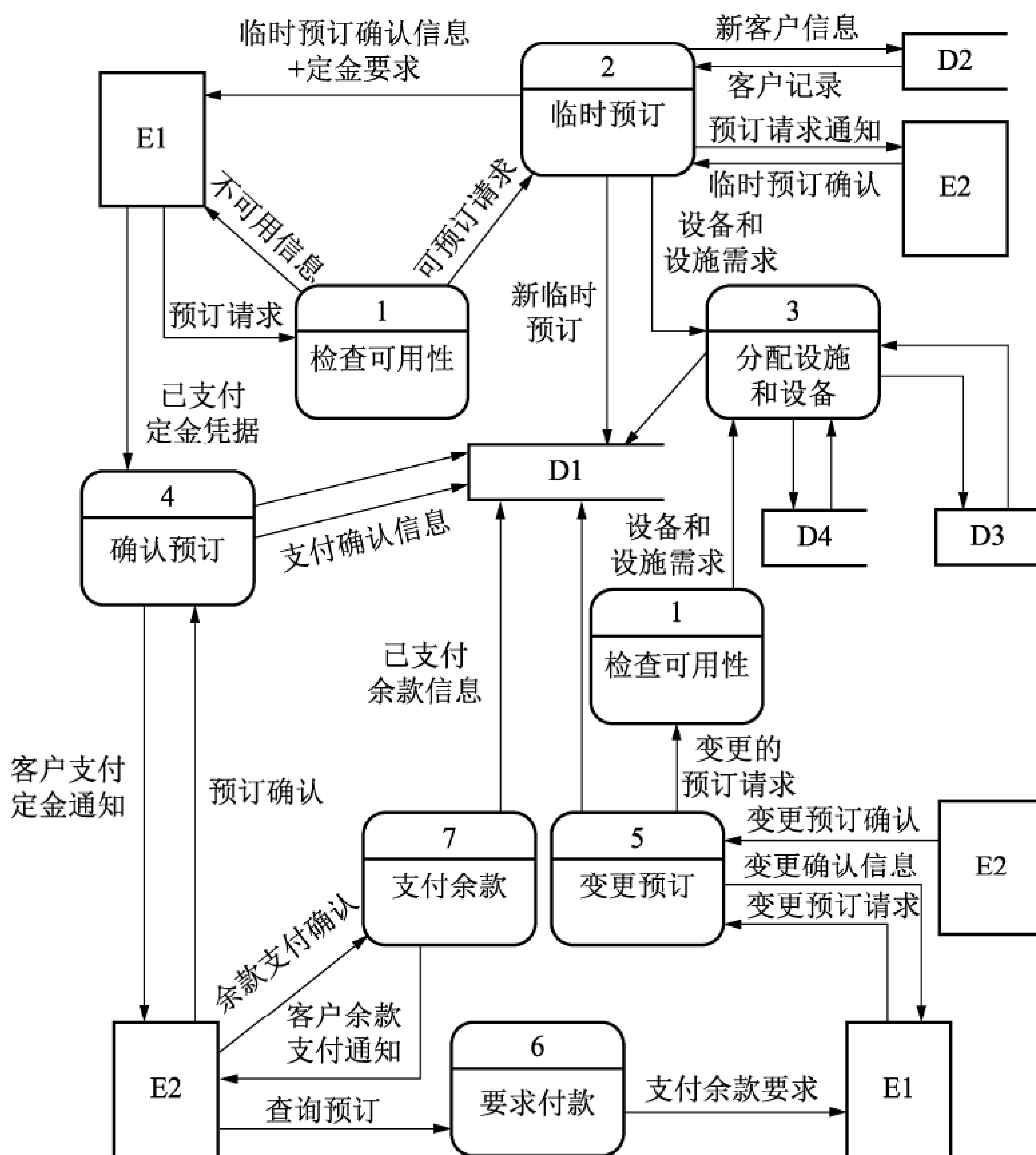


图 4-4　0 层数据流图

【问题 1】(2 分)

使用说明中的词语，给出图 4-3 中的实体 E1~E2 的名称。

【问题 2】(4 分)

使用说明中的词语，给出图 4-4 中的数据存储 D1~D4 的名称。

【问题 3】(6 分)

根据说明和图中术语，补充图 4-4 之中缺失的数据流及其起点和终点。

【问题 4】(3 分)

如果发送给客户的确认信息是通过 E-mail 系统向客户信息中的电子邮件地址进行发送的，那么需要对图 4-3 和图 4-4 进行哪些修改？用 150 字以内文字加以说明。

解析：

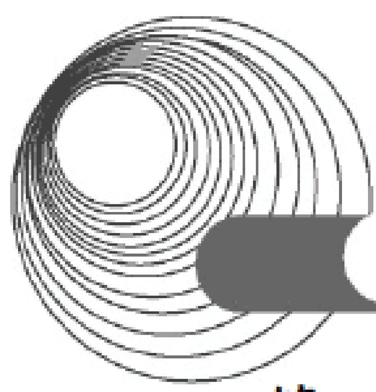
该题以会议预订系统来考查学生对数据流图知识点的掌握程度。从题目的问答形式上来看和往年相似。

【问题 1】

根据 0 层数据流中 E1 向系统发送预订请求数据流可知，E1 实体为“客户”；从预订请求通知到临时预订确认可知 E2 实体为“管理员”。

【问题 2】

根据题目对功能的描述，结合 0 层数据流图，新临时预订提交、变更的预订请求提交



等，可知 D1 为预订表；新客户信息存入 D2 中，可知 D2 为客户信息记录表；根据分配设施和设备数据流，可以得到 D3、D4 分别为设施表和设备表。

【问题 3】

由“确认预订”收到客户支付定金的通知后，检查确认更新预订表，同时要向客户发送预订确认信息，存在一个起点为“确定预订点”到终点为 E1 的数据流，即预订确认信息数据流；根据临时预订描述，首先要由客户发送预订请求，提交确认，系统生成新临时预订存入预订表，所以存在一个起点为客户即 E1，终点为“临时预订”的数据流，即客户临时预订信息数据流。

【问题 4】略。

答案：

【问题 1】

E1：客户

E2：管理员

【问题 2】

D1：预订表

D2：客户信息记录表

D3：设施表

D4：设备表

【问题 3】

数据流名称：预订确认信息　起点：确定预订点终点：E1

数据流名称：客户信息　起点：E1　终点　临时预订

【问题 4】

图 4-5 中：增加外部实体“第三方 E-mail 系统”，将临时预订/预订/变更确认信息终点均修改至“第三方 E-mail 系统”。

图 4-6 中：增加外部实体“第三方 E-mail 系统”，增加加工“发送邮件”，将临时预订/预订/变更确认信息终点均修改至“发送邮件”加工，并增加从 D2 到“发送邮件”加工的数据流“电子邮件地址”，再从发送邮件加工引出数据流“临时预订/预订/变更确认信息”，其终点为“第三方 E-mail 系统”。

4.1.3　同步练习

1. 阅读以下说明和图 4-5，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

有一种游戏，其规则如下：有一个 3×3 的方格，每个方格中只可画“+”符号或“−”符号，表示该方格的值。图 4-5(a)定义了各方格的位置，表 4-1 为每个方格位置定义了与其相关联的位置集，各方格的初值如图 4-5(b)所示。游戏开始后，每次可选一个值为“+”的方格位置，然后根据表 4-1 将该位置所对应的每个相关联的位置上的符号重画成与其不同的

符号，即将“+”重画成“−”，将“−”重画成“+”。重画操作可用所选的位置编号来描述。例如，在如图 4-5(b)所示的情况下，选择位置 4 时，重画结果如图 4-5(c)所示。经过连续的若干次这样的操作后，当 3×3 方格呈现出如图 4-5(d)所示的图形时，表示获胜；当呈现出如图 4-5(e)所示的图形时，表示失败。

```
0 1 2      − − −      − + −      + + +      − − −
3 4 5      − + −      + − +      + − +      − − −
6 7 8      − − −      − + −      + + +      − − −
 (a)        (b)        (c)        (d)        (e)
```

图 4-5　游戏示例图

表 4-1　方格位置及其相关位置集对照表

方格位置	相关位置
0	0 1 3 4
1	0 1 2
2	1 2 4 5
3	0 3 6
4	1 3 4 5 7
5	2 5 8
6	3 4 6 7
7	6 7 8
8	4 5 7 8

图 4-6 所示的流程图旨在输出从初始状态出发直至获胜的重画操作(即所选的位置编号)序列。图中假定数组 A[0..8]存放 3×3 方格的值，数组 c[0..8][1..5]存放表 4-1 所示的各方格位置的相关联的位置集，数组 d[0..8]存放各方格位置的相关联的位置个数，数组元素 S[1]~S[k]存放各次重画操作所对应的位置编号，变量 N 存放 3×3 方格中当前“+”符号的个数。

【问题 1】

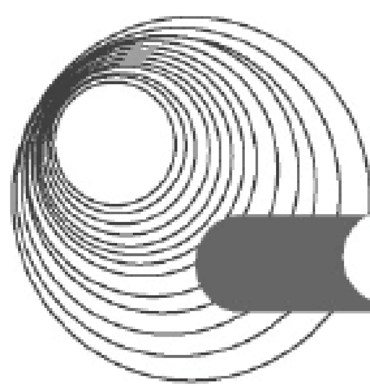
填充图 4-6 中的(1)~(4)空。

【问题 2】

图 4-6 中的(5)应与 A、B、C 中的哪一点连接？

【问题 3】

如果每次由游戏者选择方格改由程序自动枚举选择，那么从初态出发求出所有可能的获胜重画操作序列，在哪些情况下需要进行回溯处理？



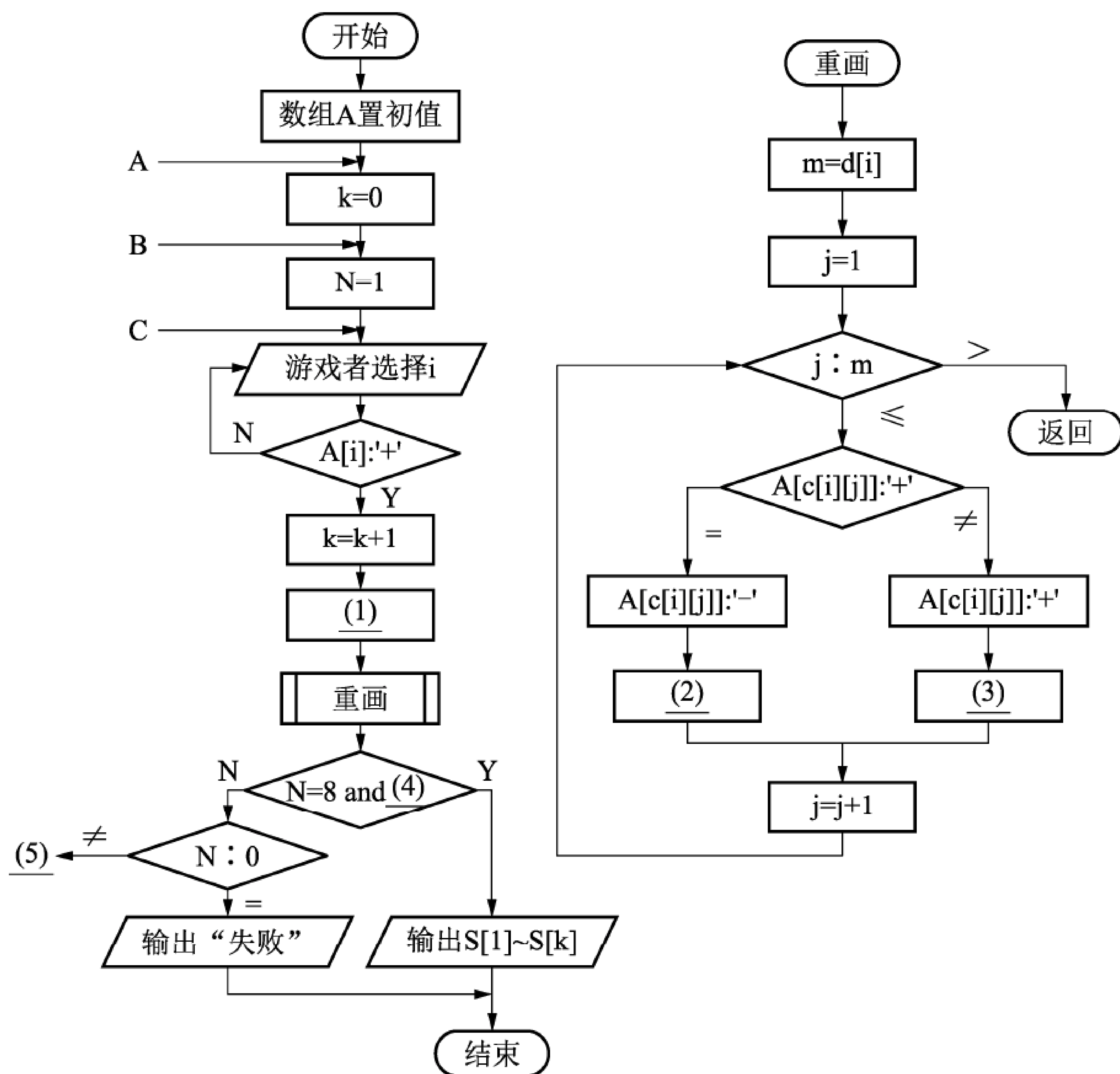
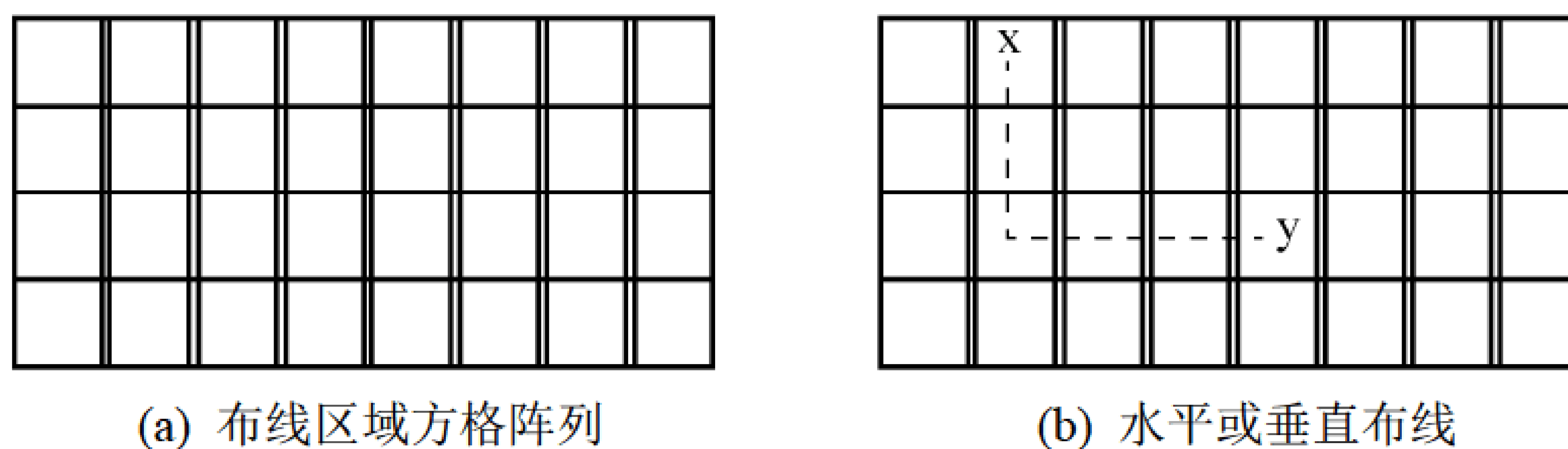


图 4-6 流程图

2. 阅读以下说明和图，从供选择的答案中选出应填入流程图中(1)~(5)处的子句，写在答题纸的对应栏内。

【说明】

一个印制电路板的布线区域可分成 n×m 个方格，如图 4-7(a)所示，现在需要确定电路板中给定的两个方格的中心点之间的最短布线方案。电路只能沿水平或垂直方向布线，如图 4-7(b)中虚线所示。为了避免线路相交，应将已布过线的方格做成封锁标记，其他线路不允许穿过被封锁的方格。



(a) 布线区域方格阵列 (b) 水平或垂直布线

图 4-7 布线

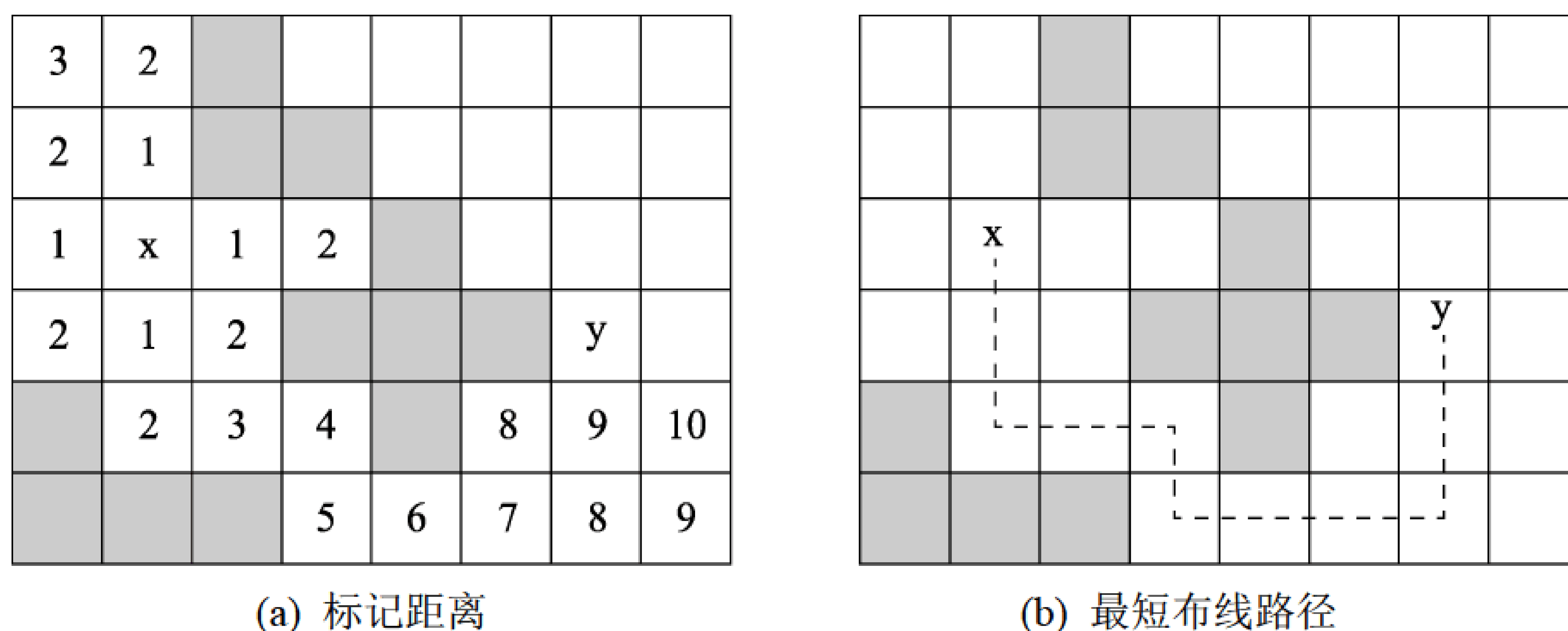
设给定印制电路板的起始方格 x 与目的方格 y 尚未布线，求这两个方格间最短布线方案的基本思路是：从起始方格 x 开始，先考查距离起始方格为 k 的某一个可达方格就是目

标方格 y 时为止，或者由于不存在从 x 到 y 的布线方案而终止。布线区域中的每一个方格与其相邻的上、下、左、右 4 个方格之间的距离为 1，依次沿下、右、上、左这 4 个方向考查，并用一个队列记录可达方格的位置。表 4-2 给出了沿这 4 个方向前进 1 步时相对于当前方格的相对偏移量。

表 4-2　沿 4 个方向前进 1 步时相对于当前方格的相对偏移量

搜索顺序 i	方　向	行偏移量	列偏移量
0	上	−1	0
1	右	0	1
2	上	−1	0
3	左	0	−1

例如，设印制电路板的布线区域可划分为一个 6×8 的方格阵列，如图 4-8(a)所示，其中阴影表示已封锁方格。从起始方格 x(位置[3, 2]，标记为 0)出发，按照下、右、上、左的方向依次考查，所标记的可达方格如图 4-8(a)所示，目标方格为 y(位置[4, 7]，标记为 10)，相应的最短布线路径如图 4-8(b)中虚线所示。



(a) 标记距离　　(b) 最短布线路径

图 4-8　标记距离与最短布线路径

图 4-9 和图 4-10 所示的流程图即利用上述思路在电路板方格阵列中进行标记，图中使用的主要符号如表 4-3 所示。在图 4-9 中，设置电路板初始格局，即将可布线方格置为数值 −1、已布线方格(即封锁方格)置为−9。设置方格阵列“围墙”的目的是省略方格位置的边界条件判定，方法是在四周附加格，并将其标记为−9(与封锁标记相同)。

供选择的答案：

[a] Found ≠ true　　[b] Found = true
[c] T = EndPos　　[d] Q.insert(T)
[e] T ← Q.delete()　　[f] CurPos = EndPos
[g] i≥4　　[h] CurPos ← Q.delete()
[i] Grid[T.row,T.col] = −1　　[j] Grid[T.row,T.col] ≠ −1

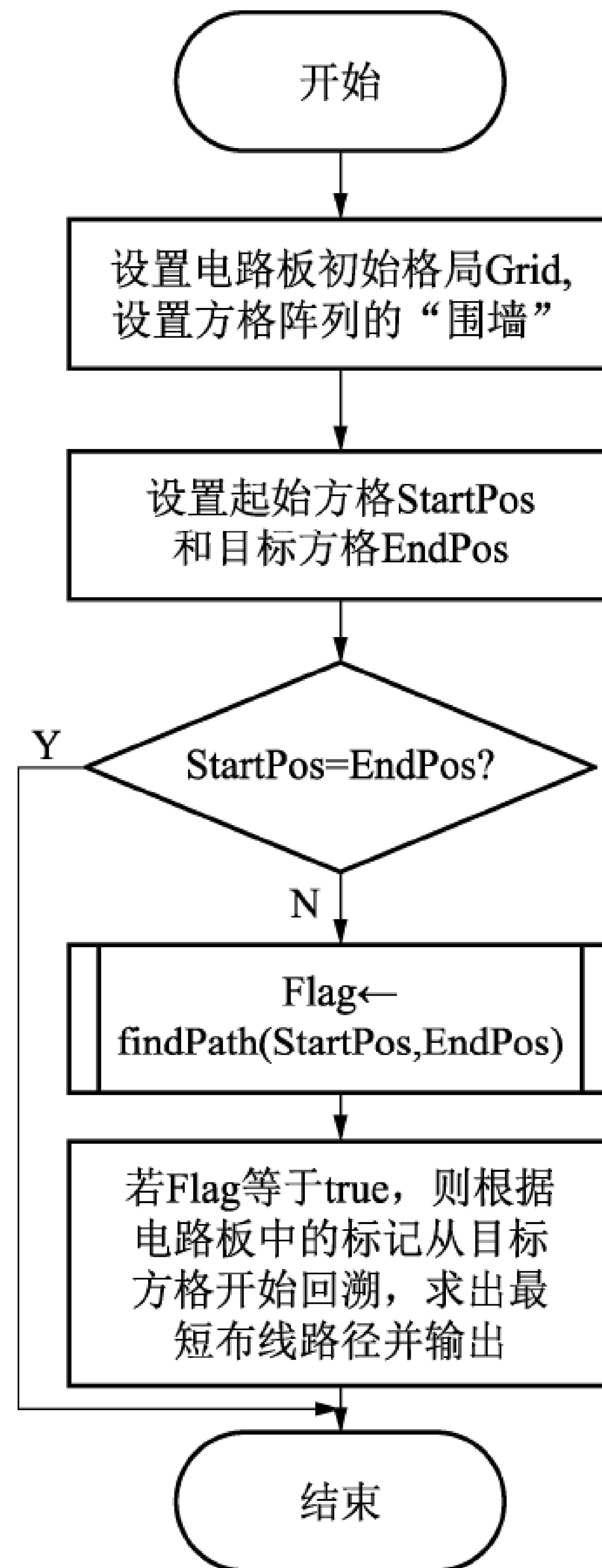


图 4-9　流程图(一)

表 4-3　图中使用的主要符号

符　号	含　义
Grid	全局二维数组 Grid[N+2,M+2]表示电路板方格阵列，初始时数组元素 Grid[i,j]的值为−1 时，表示当前方格可布线；为−9 时，表示当前方格不可布线
Offset	一维数组 Offset[4]:Offset[i](0≤i≤3)的分量为 r(行偏移量)和 c(列偏移量)，按照表 4-2 的内容设置其值
StartPos、EndPos、CurPos、T	分别表示起始方格、目标方格、当前方格和临时方格，其位置用分量度 row 和 col 确定
Q.insert(s)	将方格 s 的位置信息加入队列
Q.delete()	删除非空队列的队头元素，并返回该元素
Q.empty()	若队列 Q 为空，则返回 true；否则返回 false

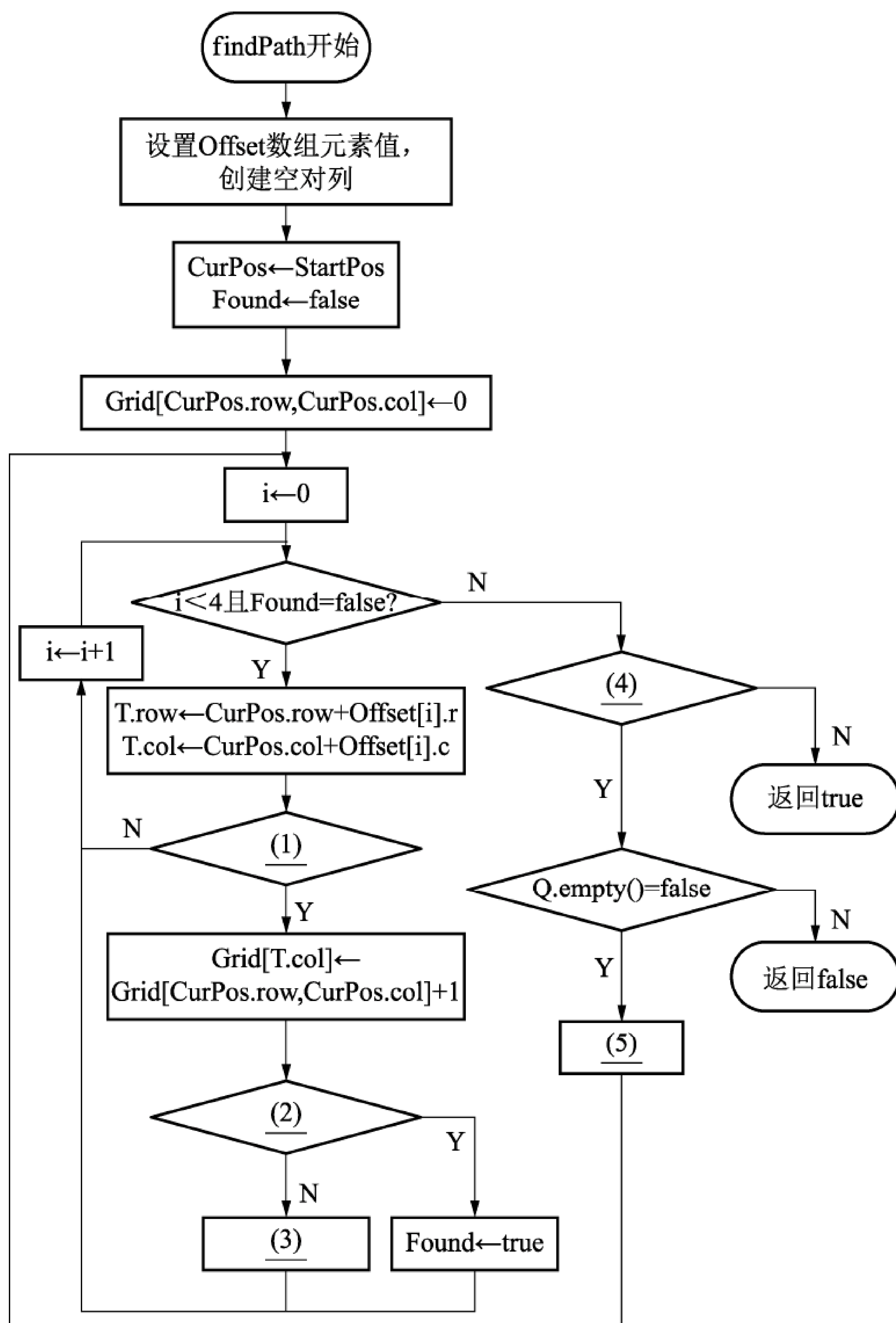


图 4-10　流程图(二)

3. 阅读下列算法说明和图 4-11，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

某旅馆共有 N 间客房。每间客房的房间号、房间等级、床位数及占用状态分别存放在数组 ROOM、RANK、NBED 和 STATUS 中。房间等级值为 1、2 或 3。房间的状态值为 0(空闲)或 l(占用)。客房是以房间(不是床位)为单位出租的。

本算法根据几个散客的要求预订一间空房。程序的输入为人数 M，房间等级要求 R(R=0 表示任意等级都可以)。程序的输出为所有可供选择的房间号。

【问题 1】

假设当前该旅馆各个房间的情况如表 4-4 所示。当输入 M=4，R=0 时，该算法的输出是什么?

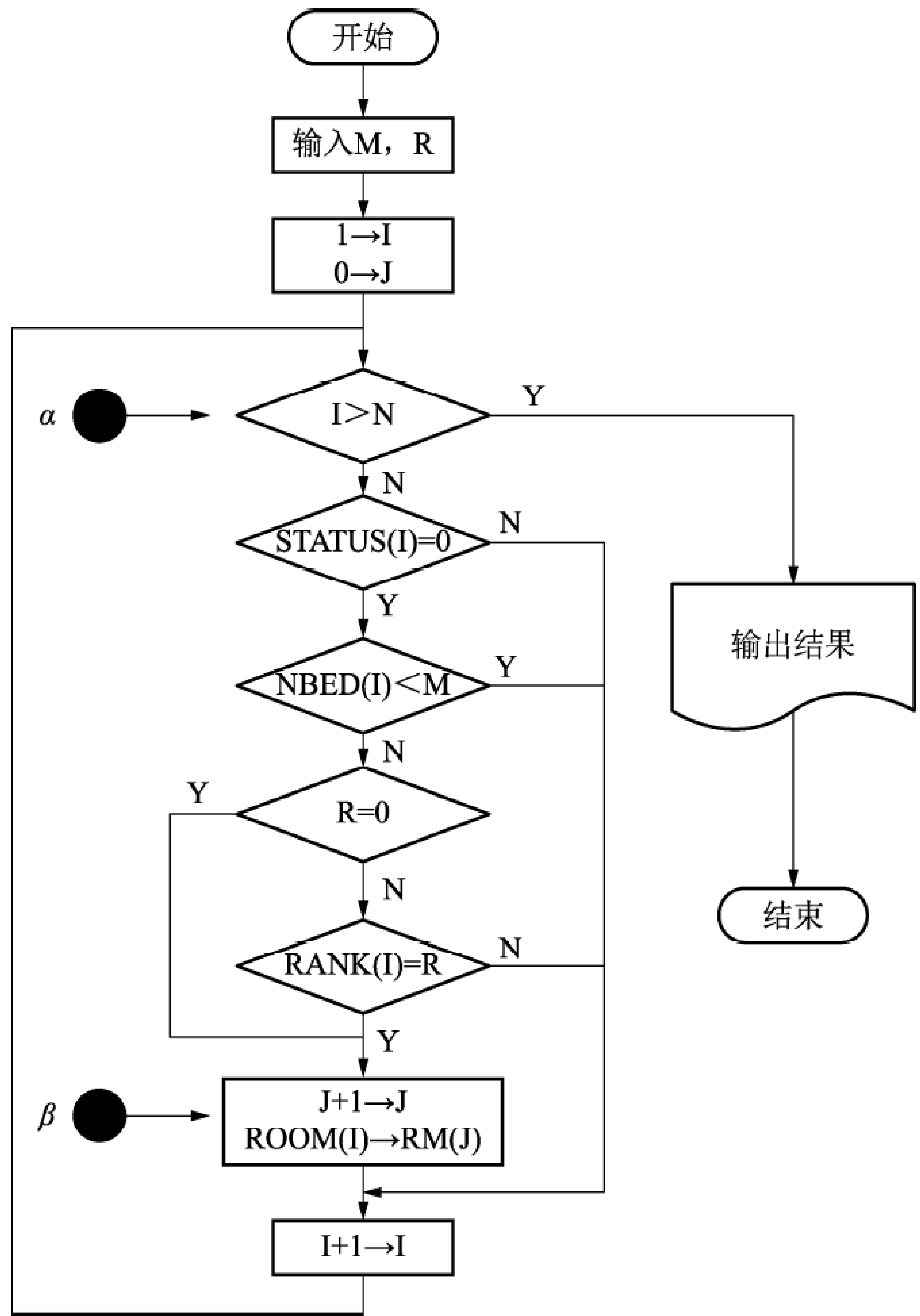


图 4-11　算法流程图

【问题 2】

如果等级为 R 的房间每人每天的住宿费为 RATE(R)，RATE 为数组。为使该算法在输出每个候选的房间号 RM(J)后，再输出这批散客每天所需的总住宿费 DAYRENT(J)，图 4-11 中β所指框中的最后处应增加什么处理?

【问题 3】

如果限制该算法最多输出 K 个可供选择的房间号，则在图 4-11 中α所指的判断框应改成什么处理?

表 4-4　当前该旅馆各个房间的情况

序号 i	ROOM	RANK	NBED	STATUS
1	101	3	4	0
2	102	3	4	1
3	201	2	3	0
4	202	2	4	1
5	301	1	6	0

4.1.4　同步练习参考答案

1.

【问题 1】

(1) S[k] = i。(2) N = N−1。(3) N = N+1。(4) A[4] = '−'。

【问题 2】

(5) 应与 C 相连。

【问题 3】

在以下情况下需要进行回溯处理。

(1) 获胜。

(2) 失败。

(3) 出现了以前出现过的图形。

(4) 一种图形的枚举选择完。

2.

(1) [i]或 i。

(2) [c]或 c。

(3) [d]或 d。

(4) [a]或 a。

(5) [h]或 h。

3.

【问题 1】

101，301。

【问题 2】

RATE(RANK(I)) * M → DAYRENT(J) 或 M * RATE(RANK(I)) → DAYRENT(J)。

【问题 3】

I > N OR J=K。

其中，I > N 也可以写成 I = N+1；J = K 也可以写成 J≥K。

4.2　本 章 小 结

本章知识点在 2009 年的新大纲中改动不大。

根据近几年软件设计师水平考试试题分布情况来看，程序流程图不是每年都考，但仍是大纲要求的内容。每次如果考到，占的分数也是一定的，几个填空，一共 15 分。

程序流程图只是算法或者程序的一种表示形式，实际上考查的还是考生的算法设计能力。所以，只要考生理解算法，结合平时的编程功夫，就可以正确补充流程图中缺少的语句。

第 5 章　算 法 设 计

大纲要求：

- 掌握 C 程序设计语言，能够进行编程和测试，并进行必要的优化。
- 掌握常用数据结构及其常用操作。
- 掌握排序算法、查找算法。
- 灵活应用各种算法设计策略。

5.1　算法设计的基础知识

5.1.1　考点辅导

5.1.1.1　指针

指针是 C 语言中最为重要也是最难的一个关键点，很多数据结构都是基于指针实现的，如链表、链式队列、链栈、二叉树等。

所谓指针，就是一个用来存储地址的变量。这可谓指针的本质，需要牢记。

也许你会很纳闷，指针为什么一定要定义成某类型(int、char)呢？指针不能就是“指针类型”吗？接触过汇编的人就很容易理解为什么。存储单元的单位是字节，就是说一般地址是按字节编址的，对一个地址进行操作(读取或赋值)就要指明是对单字节(不用特别声明)、两字节(WORD PTR)，还是双字节(四字节，DWORD PTR)进行操作。同样，指针是存储地址的，即指针就是一个地址，自然也要说明其类型；而且，这个类型还关乎指针自加自减时真正加减的字节数。

顺便说一下，数组名也是指针。数组在申请空间时，数组名存储该存储空间的首地址。注意：数组名存储的是地址，因此也是指针，只是该指针一旦赋值后就不能修改，即所谓常指针。当直接输出数组名时，输出的其实是数组的首地址。这样，当形参声明为指针时，亦可将数组名作为实参进行传递。因此，可以用指针的方式访问数组中的元素，如下例采用指针的方式遍历输出数组。

```
#include <stdio.h>
int main()
{
    int array3[6]={100,101,102,103,110,111};
    /***********指针写法，c 中是按行优先存储的************/
    printf("%x\n", array3);//输出首地址，十六进制输出
    int *p;
    p = array3;             // 注意，一维数组名相当于指针
    while(p <= &array3[5]){// &array3[5]表示最后一个元素的地址
```

```
        printf("%8d", *(p++));
    }
    printf("\n");
    return 0;
}
```

当指针作为函数参数传递时，需要特别注意 C 语言中的“值”传递原则。下例中的函数希望为指针 p 申请空间，但不能达到目的，为什么呢？

```
void GetMemory(char *p)
{
    char *s=NULL;
    s=(char*)malloc(100*sizeof(char));
    p=s;
}
```

归根结底，C 函数的形参与实参之间只是“值传递”：当形参是普通变量时，传递的是实参的值；当形参是指针时，传递的是指针变量的值，即某变量的地址，这样可以通过指针成功地改变其所指单元的值，但自身的改变不会传回给实参。上例可改为：

```
void GetMemory(char **p)
{
    char *s=NULL;
    s= (char*)malloc(100*sizeof(char));
    *p=s;
}
```

注意：这样修改后，调用时实参应该是指针的“地址”(或指向指针的指针)。这样即上面所说的可以改变指针所指单元的值，因此可达到预期目的。

5.1.1.2　常用数据结构

树(特别是二叉树)和栈相对较为重要，要重点掌握。

1. 线性表

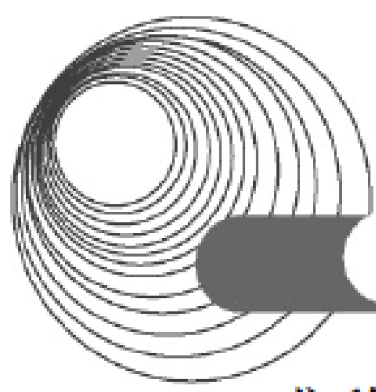
1) 概念

在数据结构中，线性结构通常称为线性表，它是最简单、最常见的一种数据结构。通常线性表是由 n 个相同数据类型的节点组成的有限序列。其特点是在数据元素的非空有限集合中，存在唯一的一个被称为“第一个”的数据元素；存在唯一的一个被称为“最后一个”的数据元素；除第一个之外，集合中的每个数据元素均有且仅有一个前驱；除最后一个之外，集合中的每个数据元素均有且仅有一个后继。

一个由 n 个节点 $e_0, e_1, \cdots, e_{n-1}$ 组成的线性表记为($e_0, e_1, \cdots, e_{n-1}$)。线性表的节点个数称为线性表的长度，长度为 0 的线性表称为空的线性表，简称空表。对于非空线性表，e_0 是线性表的第一节点，e_{n-1} 是线性表的最后一个节点。线性表的节点构成了一个序列，对序列中两个相邻节点 e_i 和 e_{i+1}，称前者为后者的前驱节点，后者是前者的后继节点。

2) 重要性质

线性表最重要的性质是线性表中节点的相对位置是确定的。线性表的节点也称为表元，



或称为记录，要求线性表的节点一定是同一类型的数据。线性表的节点可由若干个成分组成，其中唯一标识表元的成分称为关键字，简称键。

3) 基本操作

线性表的基本操作如下。

- Initiate(L)：初始化操作，对线性表中的各个元素的初始值进行初始化。
- Length(L)：求长度函数，线性表的长度就是线性表中元素的个数。
- Get(L, i)：取元素函数，取线性表中的一个元素。
- Prior(L, elm)：求前驱函数，给定一个线性表元素，求出其前一个元素。
- Next(L, elm)：求后继函数，给定一个线性表元素，求出其后一个元素。
- Locate(L, x)：定位函数，根据给定的值，在线性表中找到并返回其位置。
- Insert(L, i, b)：插入操作，给定一个元素后，将其插在线性表的某个位置。
- Delete(L, i)：删除操作，删除给定位置或值的元素。

4) 存储结构

有多种存储方式能将线性表存储在计算机内，其中最常用的是顺序存储和链式存储。根据存储方式的不同，其上述基本操作的实现也不一样。

(1) 顺序存储：顺序存储是最简单的存储方式，其特点是逻辑关系上相邻的两个元素在物理位置上也相邻。通常使用一个足够大的数组，从数组的第一个元素开始，将线性表的节点依次存储在数组中。顺序存储方式的优点是能直接访问线性表中的任意节点。线性表的第 i 个元素 a[i]的存储位置可以使用以下公式求得：$Loc(a_i)=Loc(a_1)+(i-1)\times 1$。式中 $Loc(a_1)$ 是线性表的第一个元素 a_1 的存储位置，通常称作线性表的起始位置或基地址。顺序存储的缺点是：线性表的大小固定，浪费大量的存储空间，不利于节点的增加和减少；执行线性表的插入和删除操作要移动其他元素，不够方便。

(2) 链式存储：链式存储是用链表来存储线性表。其特点是：每个链表都有一个头指针，整个链表的存取必须从头指针开始，头指针指向第一个数据元素的位置，最后的节点指针为空。当链表为空时，头指针为空值。当链表非空时，头指针指向第一个节点。链式存储的缺点是：由于要存储地址指针，所以浪费空间；直接访问节点不方便。

链式存储有单链表(线性链表)、循环链表、双向链表。

- 单链表从链表的第一个表元开始，将线性表的节点依次存储在链表的各表元中。链表的每个表元除要存储线性表节点信息外，还要存储其后继节点的指针。
- 循环链表是单链表的变形，其特点是表中最后一个节点的指针域指向头节点，整个链表形成一个环。因此，从表中的任意一个节点出发都可以找到表中的其他节点。循环链表中，从头指针开始遍历的结束条件不是节点的指针是否为空，而是是否等于头指针。为简化操作，循环链表中往往加入表头节点。
- 双向链表的节点中有两个指针域，一个指向直接后继，另一个指向直接前驱，克服了单链表的单向性缺点。

2. 队列

1) 概念

队列(Queue)是一种先进先出(FIFO)的线性表，队列是只允许在一端进行插入操作，另一端进行删除操作的线性表。允许删除的那一端称为队首(Front)，允许插入的那一端称为队

尾(Rear)。通常称队列的节点的插入为进队，队列的节点的删除为出队。若有队列 $Q=(q_0, q_1, \cdots, q_{n-1})$，则 q_0 称为队首节点，q_{n-1} 称为队尾节点。

2) 存储结构

可以用顺序存储线性表来表示队列，也可以用链表来实现，用链表实现的队列称为链队列。

3) 优先级队列

优先级队列是一种不同于先进先出队列的另一种队列，每次出队的是队列中最高优先级的元素。

3. 栈

1) 概念

栈(Stack)是限定仅在表尾进行插入或删除操作的线性表。表尾端称为栈顶(Top)，表头端称为栈底(Bottom)。故栈是后进先出(LIFO)的线性表。

若有栈 $S=(S_0,S_1,\cdots,S_{n-1})$，则 S_0 称为栈底节点，S_{n-1} 称为栈顶节点。通常称栈的节点的插入为进栈(Push)，栈的节点的删除为出栈(Pop)。

2) 存储结构

栈有两种存储结构：顺序栈和链栈。

- 顺序栈即栈的顺序存储结构，是利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素，同时设指针 top 指示栈顶元素的当前位置。
- 链栈即栈的链式存储结构，链表的第一个元素是栈顶元素，链表的末尾是栈底节点，链表的头指针就是栈顶指针，栈顶指针为空则是空栈。

4. 堆

1) 定义

n 个元素的序列 $\{k_1, k_2,\cdots, k_n\}$ 当且仅当满足以下的关系式时才称之为堆：$\begin{cases}k_i \leqslant k_{2i} \\ k_i \leqslant k_{2i+1}\end{cases}$ 或 $\begin{cases}k_i \geqslant k_{2i} \\ k_i \geqslant k_{2i+1}\end{cases}$，并相应地称为小顶堆或大顶堆。

2) 判断办法

判断堆的办法是把序列看成一棵完全二叉树，若树中所有非终端节点的值均不大于(或不小于)其左右孩子的节点的值，则该序列为堆。

3) 典型应用

堆的典型应用是堆排序。堆排序首先要根据待排序记录的关键字建立初始堆，其方法是：将待排序的关键字按层序遍历方式分放到一棵完全二叉树的各个节点中，显然所有 $i>[n/2]$ 的节点 k_i 都没有子节点，以这样的 k_i 为根的子树已经是堆，因此初始堆可从完全二叉树的第$(i=[n/2])$个节点开始，通过调整，逐步使以 $k_{[n/2]}$，$k_{[n/2]-1}$，⋯，k_2，k_1 为根的子树满足堆的定义。

注意：堆与一棵完全二叉树对应，但堆本身是线性表。

5. 数组

数组是最常用的数据结构之一，在程序中，数组常用来实现顺序存储的线性表。数组由固定个数的元素组成，全部元素的类型相同，元素依次顺序存储。每个元素对应一个下标，数组元素按数组名和元素的下标引用，引用数组元素的下标个数称为数组的维数。

在 C 语言中，n 个元素的数组中，第一个元素的下标为 0，最后一个元素的下标为 n−1。

数组可以分为静态数组和动态数组两类。

(1) 静态数组是指数组的存储空间分配是在使用之前进行，在程序运行中不能改变，不利于数组的扩展。

(2) 动态数组是在程序执行中进行数组存储空间的分配。

动态数组一般采用链式存储结构，而静态数组一般采用顺序存储结构。

数组元素可以是任意类型，当元素本身又是数组时，就构成了多维数组。多维数组是一维数组的推广，最常用的是二维数组。在 C 语言中，数组元素按行优先顺序存放。

一般用多维数组表示矩阵，矩阵的类型有对称矩阵、三角矩阵(下三角矩阵或上三角矩阵)和对角矩阵。

稀疏矩阵的存储：用顺序存储结构的三元数组对稀疏矩阵进行存储，分别记录行、列和值。

6. 树

1) 定义

树型结构是一类重要的非线性数据结构，其中以树和二叉树最为常用。

树是由一个或多个节点组成的有限集 T，它满足以下两个条件：有一个特定的节点称为根节点；其余的节点分成 m 个互不相交的有限集 $T_1, T_2, \cdots, T_m$，其中每个集又都是一棵树，称 $T_1, T_2, \cdots, T_m$ 为根节点的子树。

可见树的定义是递归的，即一棵树由子树构成，子树又由更小的子树构成。

2) 相关概念

- 一个节点的子树数目称为该节点的度。
- 树中各节点的度的最大值称为树的度。
- 树中节点的最大层次称为树的深度。

若将树中节点的各子树看成是从左到右具有次序的，即不能交换，则称该树为有序树，否则称为无序树。

3) 树的遍历

在应用树结构时，常要求按某种次序获得树中全部节点的信息，这可通过树的遍历操作来实现，常用的遍历方法如下。

- 前序：先访问根节点，然后从左到右遍历根节点的各棵子树。
- 后序：先从左到右遍历根节点的各棵子树，然后访问根节点。
- 层序：先访问处于 1 层上的节点，然后从左到右依次访问处于 2 层、3 层上的节点，即从上到下、从左到右逐层访问树中各层上的节点。

7. 二叉树

1) 定义

二叉树是 n 个节点的有限集合，它或者是空树，或者是由一个根节点及两棵不相交的、分别称为左子树和右子树的树所组成。

2) 树和二叉树最主要的区别

二叉树的节点的子树要区分左子树和右子树，即使在节点只有一棵子树的情况下也要明确指出该子树是左子树还是右子树；另外，二叉树的节点的最大度为 2，而树中不限制节点的度数。

3) 二叉树的性质

- 在二叉树的第 i 层至多有 2^{i-1} 个节点(根节点为 1 层)。
- 深度为 k 的二叉树至多有 2^k-1 个节点。
- 对任何一棵二叉树 T，如果其终端节点数为 n_0，度为 2 的节点数为 n_2，则 $n_0=n_2+1$。
- 具有 n 个节点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$。

4) 特殊二叉树

- 满二叉树：二叉树上每一层的节点数目达到最大值。
- 完全二叉树：除了最外层，其余层上的节点数目都达到最大值，而第 h 层上的节点集中存放在左子树中。
- 平衡二叉树(AVL 树)：二叉排序树的一种，其检索效率介于最优二叉树(Huffman 树)和一般二叉排序树之间，要求任何一个节点的左右子树的高度差都不大于 1。
- 二叉查找树：又称二叉排序树，左子树的值都小于根节点的值，而右子树的值都大于根节点的值，同时左、右子树都是查找树。

5) 二叉树的遍历

二叉树的遍历是非常重要的操作，有前序、中序、后序和层序遍历，要求熟练掌握。

6) 二叉树的非递归遍历

二叉树因具有递归性质，因此能很方便地用递归实现遍历，借助栈同样可以用非递归方式实现遍历。注意体会栈的使用，在此采用不带头节点的单链表作为栈的存储结构，Stop 是栈顶指针，注意出栈/入栈操作。

C 代码如下：

```
typedef struct BiTnode{//二叉树节点结构
    int data;
    struct BiTnode *lchild, *rchild;
}BiTnode, *BiTree;
typedef struct SNode{//链栈的节点结构
    BiTree elem;      //栈中的元素是指向二叉链表节点的指针
    struct SNode *next;
}SNode;
int InOrderTraverse(BiTree root)
{
    BiTree p;
    SNode *q, *Stop = NULL;//用不带头节点的单链表作为栈的存储结构
    p = root;
```

```
    while(p != NULL || Stop != NULL){
        if(p != NULL){//不是空树
            q = (SNode*)malloc(sizeof q);
            if(q == NULL) return -1;
            q→next = Stop; q→elem = p;
            Stop = q;//根节点入栈
            p = p→lchild;//进入根的左子树
        }else{
            q = Stop; Stop = Stop→next;//栈顶元素出栈
            printf("%d", q→elem→data);//访问根节点
            p = q→elem→rchild;//进入根的右子树
            free(q);//释放原栈顶元素的节点空间
        }/*if*/
    }/*while*/
    return 0;
}/*InOrderTranverse*/
```

8. Huffman 树

1) 定义

Huffman 树又称为最优二叉树，是一类带权路径长度最短的树。Huffman 树是指权值为 w_1，w_2，…，w_n 的 n 个叶子节点的二叉树中带权路径长度最小的二叉树。

2) 相关概念

(1) 路径是指从树中一个节点到另一个节点之间的分支构成的这两个节点之间的路径，路径上的分支数目就称为路径长度。

(2) 树的路径长度是从树根到每一个叶子之间的路径长度之和。

(3) 节点的带权路径长度为从该节点到树根之间的路径长度与该节点权的乘积。树的路径长度为树中所有节点的带权路径长度之和，记为 $WPL=\sum_{k=1}^{n} w_k l_k$，其中 n 为带权叶子节点数目；$w_k$ 为叶子节点的权值；l_k 为叶子节点到根的路径长度。

3) 构造 Huffman 树的算法

(1) 给定 n 个节点的集合，每个节点都带权值。

(2) 选两个权值最小的节点构造一棵新的二叉树，新的二叉树的根节点的权值就是两个子节点权值之和。

(3) 从 n 个节点中删除刚才使用的两个节点，同时将新产生的二叉树的根节点放在节点集合中。

(4) 重复步骤(2)、步骤(3)，直到只有一棵树为止。

4) Huffman 树的应用

Huffman 树的应用是 Huffman 编码，在编码过程中要考虑两个问题，一是数据的最小冗余编码问题，二是译码的唯一性问题。在实际应用中，各个编码字符的出现频率不同，希望用最短的编码来表示出现频率大的字符，而用较长的编码表示出现频率较小的字符，从而使整个编码序列的总长度最小，这就是最小冗余编码问题。Huffman 编码就解决了这个问题，根据权值或概率的大小来构建 Huffman 树，然后左分支用 0 表示，而右分支用 1 表示，这样就形成了编码序列。

9. 图

图是一种较线性表和树更为复杂的数据结构。在图形结构中，节点之间的关系可以是任意的，图中任意两个数据元素之间都可能相关。

5.1.1.3　常用算法设计技术

常用的算法设计技术主要有迭代法、穷举搜索法、递推法、递归法、回溯法、贪心法、分治法、动态规划法等。递归法、回溯法及贪心法极为重要，要重点掌握。

1. 迭代法

迭代法是用于数值计算近似求解的一种常用的算法，确定一合适的迭代公式，选一初始近似值及解的误差，循环处理实行迭代过程，终止条件是前后两次得到的近似值之差的绝对值小于给定的误差。

设方程为 $f(x)=0$，用某种属性方法导出其等价形式 $x=g(x)$，然后按以下步骤进行。

(1) 选取一个方程的近似根，赋给变量 x_0。

(2) 将 x_0 的值保存于变量 x_1，然后计算 $g(x_1)$，并将结果保存于变量 x_0 中。

(3) 当 x_0 与 x_1 的差的绝对值还小于给定的精度要求时，重复步骤(2)的计算。

若方程有根，并且上述方法计算出来的近似根序列收敛，则按上述方法求得的 x_0 就认为是方程的根。

C 语言形式如下：

```
{
    x0 = 初始近似根;
    do{
        x1 = x0;
        x0 = g(x1);
    }while(fabs(x0-x1) > Epsilon);
    printf("方程的近似根为%f\n", x0);
}
```

具体使用迭代法求根时应注意以下两种可能的情况：如方程无解，算法求出的近似根序列就不会收敛，迭代过程会变成“死循环”，因此在使用迭代算法前应先考查方程是否有解，并在程序中对迭代的次数给予限制；方程虽然有解，但迭代公式选择不当，或迭代的初始近似根选择不合理，也会导致迭代失败。

2. 穷举搜索法

穷举搜索法也称枚举法，是对可能是解的众多候选解按某种顺序进行逐一枚举和检验，并从中找出那些符合要求的候选解作为问题的解。

要解决的问题只有有限种可能，在没有更好的算法时，总可用穷举搜索的办法解决，即逐个检查所有可能的情况。

3. 递推法

1) 基本思想

递推法是利用问题本身所具有的一种递推关系求问题解的一种方法。设要求问题规模为 N 的解，当 N=1 时，解或为已知，或能非常方便地得到解。能采用递推法构造算法的问

题有重要的递推性质，即当得到问题规模为 i-1 的解后，由问题的递推性质，能从已求得的规模为 1, 2, …, i-1 的一系列解构造出问题规模为 i 的解。这样，程序可从 i=0 或 i=1 出发，重复地，由已知 i-1 规模的解，通过递推，获得规模为 i 的解，直至得到规模为 N 的解。

2) 典型应用：Fibonacci 级数

Fibonacci 级数数列为 0, 1, 1, 2, 3, 5, 8, 13, …，即 $F(0)=0$，$F(1)=1$，…，$F(n)=F(n-1)+F(n-2)(n>1)$。递推法实现算法如下：

```
int f(int n)
{
    int f0 = 0, f1 = 1, f, i;
    if(n == 0) return 0;
    if(n == 1) return 1;
    for(i = 2; i <= n; i++){
        f = f0+f1;        //由前两步的结果得到当前结果，“推出”
        f0 = f1;          //把原来的前一步当作下一次的前两步，“传递”
        f1 = f;           //把当前结果当作下一次的前一步，“传递”
                          //进行这种传递时，要注意传递的时序，在此两个语句不能颠倒
    }
    return f;
}
```

4. 递归法

1) 基本思想

递归是设计和描述算法的一种有力的工具。

能采用递归描述的算法通常有这样的特征：为求解规模为 N 的问题，设法将它分解成一些规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模稍大问题的解。特别地，当规模 N=1 时，能直接得到解。

递归算法的执行过程分为递推和回归两个阶段。在递推阶段，把较复杂的问题的求解推到比原问题简单一些的问题的求解；在回归阶段，当获得最简单情况的解后，逐级返回，依次获得稍复杂问题的解。

2) 典型应用：Fibonacci 数列和背包问题

(1) Fibonacci 数列：Fibonacci 数列的递归算法实现如下(注意与递推实现比较)。

```
int f(int n)
{
    if(n == 0) return 0;
    else if(n == 1) return 1;
    else return f(n-1)+f(n-2);
}
```

(2) 背包问题：所谓背包问题，是指有不同价值、不同重量的物品 n 件，求从这 n 件物品中选取一部分物品的选择方案，使选中物品的总重量不超过指定的限制重量，但选中物品的价值之和最大。

典型做法是逐个考查每一件物品，对于第 i 件物品的选择考虑有两种可能。

- 考虑物品 i 被选择，这种可能仅当包含它不会超过方案总重量限制时才是可行的。选中后继续递归考虑其余物品的选择。
- 考虑物品 i 不被选择，这种可能仅当不包含物品 i 也有可能找到价值更大的方案时才是可行的。

按上述思想可得递归算法如下：

```
try(物品 i, 当前选择已经达到的重量之和 tw, 本方案可能达到的总价值 tv)
{
    /*考虑物品 i 包含在当前方案中的可能性*/
    if(包含物品 i 是可接受的){
        将物品 i 不包含在当前方案中;
        if(i < n-1) try(i+1, tw+物品 i 的重量, tv);
        else if(更好的方案)//又是一个完整的方案
            以当前方案作为临时最佳方案保存;
        恢复物品 i 不包含状态;
    }
    /*考虑物品 i 不包含在当前方案中的可能性*/
    if(不包含物品 i 仍是可考虑的){
        if(i < n-1) try(i+1, tw, tv-物品 i 的价值);
        else if(更好的方案)//又是一个完整的方案
            以当前方案作为临时最佳方案保存;
    }
}
```

5. 回溯法

1) 基本思想

回溯法也称试探法，该方法首先暂时放弃关于问题规模大小的限制，并将问题的候选解按某种顺序逐一枚举和检验。当发现当前候选解不可能是解时，就选择下一个候选解；如果当前候选解除了不满足问题规模要求外，满足所有其他要求，继续扩大当前候选解的规模，并继续试探；如果当前候选解满足包括问题规模在内的所有要求时，该候选解就是问题的一个解。

在回溯法中，放弃当前候选解、寻找下一个候选解的过程称为回溯；扩大当前候选解的规模并继续试探的过程称为向前试探。

2) 典型应用：n 皇后问题

n 皇后问题是源于国际象棋的一个问题，n 皇后问题要求在一个 n×n 格的棋盘上放置 n 个皇后，使得它们彼此不受攻击。按照国际象棋的规则，一个皇后可以攻击与之在同一行或同一列或同一条斜线上的其他任何棋子。因此 n 皇后问题等价于要求在一个 n×n 格的棋盘上放置 n 个皇后，使得任何两个皇后不能被放在同一行或同一列或同一条斜线上。

求解算法如下：

```
{
    输入棋盘大小 n;      //一个 n×n 的棋盘
    m = 0;               //从空配置开始
    good = 1;            //空配置是一个合理的情况
    do{                  //循环找解
```

```
            if(good){
                if(m == n){//找到一个解
                    输出解;
                    改变之，形成下一个候选解;
                }else{
                    扩展当前候选解至下一列;
                }
            }else{  //回溯调整
                改变之，继续寻找候选解;
            }
        }while(m != 0);
    }
```

6. 贪心法

1) 基本思想

贪心法是一种不追求最优解，只希望得到较为满意解的方法。贪心法一般可以快速得到满意的解，因为省去了为找到最优解要穷尽所有可能而必须耗费的大量时间。贪心法常以当前情况为基础做出最优选择，而不考虑各种可能的整体情况，所有贪心法不需要回溯。

2) 典型应用：装箱问题

装箱问题可简述如下：设有编号为 0, 1, …, n−1 的 n 种物品，体积分别为 v_1，v_2，…，v_{n-1}，将这 n 种物品装到容量都为 V 的若干箱子里，约定这 n 种物品的体积均不超过 V。不同的装箱方案所需要的箱子数目可能不同，装箱问题要求使装尽这 n 种物品的箱子数最少。

算法描述如下：

```
{
    输入箱子的容量 V;
    输入物品种类 n;
    按体积从大到小排列，输入各物品的体积;
    预置已用箱子链为空;
    预置已用箱子计数器 box_count 为 0;
    for(i = 0; i < n; i++){//物品 i 按以下步骤装箱
        从已用的第一只箱子开始顺序寻找能放入物品 i 的箱子 j;
        if(已用箱子都不能再放物品 i){
            另用一只箱子，并将物品 i 放入该箱子;
            box_count++;
        }else{
            将物品 i 放入箱子 j 中;
        }
    }
}
```

7. 分治法

1) 基本思想

分治法也许是最广泛使用的算法设计方法，其基本思想是把大问题的解分解成一些较小的问题，然后由小问题的解方便地构造出大问题的解。

2) 典型应用：Hanoi 塔问题

Hanoi 塔问题描述如下：有 n 个盘子在 A 处，盘子从大到小，最上面的盘子最小。现在要把这 n 个盘子从 A 处搬到 C 处，可以在 B 处暂存，但任何时候都不能出现大的盘子压在小的盘子上面的情况。

当只有一个盘子时，直接从 A 移到 C 即可；如果已知 n−1 个盘子的移动方案，那么 n 个盘子的移动方案如下：先把前 n−1 个盘子从 A 借助 C 移动到 B 处，再把第 n 个盘子从 A 处直接移动到 C 处，然后再将 B 处的 n−1 个盘子从 B 处借助 A 处移动到 C 处，至此就完成了全部盘子的移动。具体 C 代码实现如下：

```
void Hanoi(int n, char a, char b, char c)//将 n 个盘子从 a 通过 b 移动到 c
{
    if(n > 1){
        Hanoi(n-1, a, c, b);     //先将前 n-1 个盘子从 a 处通过 c 移动到 b 处
        move(n, a, c);           //将第 n 个盘子从 a 处直接移动到 c 处
        Hanoi(n-1, b, a, c);     //再将前 n-1 个盘子从 b 处通过 a 移动到 c 处
    }else{
        move(n, a, c);           //只有一个盘子时，直接从 a 移动到 c。递归出口
    }
}
```

8. 动态规划法

动态规划法的基本思想是：将大问题分解成小问题，为了节约重复求相同子问题的时间，引入一个数组，不管它们是否对最终解有用，把所有子问题的解存于该数组中。

5.1.2　典型例题分析

例 1　阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2013 年 5 月试题四)

【说明】

设有 m 台完全相同的机器运行 n 个独立的任务，运行任务 i 所需要的时间为 t_i，要求确定一个调度方案，使得完成所有任务所需要的时间最短。

假设任务已经按照其运行时间从大到小排序，算法基于最长运行时间作业优先的策略：按顺序先把每个任务分配到一台机器上，然后将剩余的任务一次放入最先空闲的机器。

【C 代码】

下面是算法的 C 语言实现。

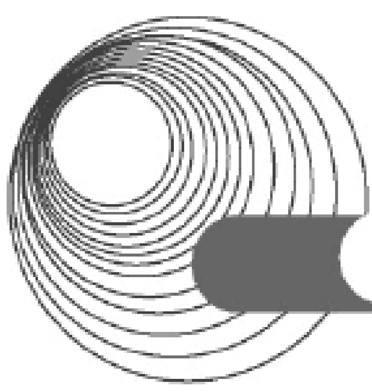
(1) 常量和变量说明。

m：机器数。

n：任务数。

t[]：输入数组，长度为 n，其中每个元素表示任务的运行时间，下标从 0 开始。

s[ ][ ]：二维数组，长度为 m×n，其中元素 s[i][j]表示机器 i 运行的任务 j 的编号，下标从 0 开始。



d[ ]：数组，长度为 m，其中元素 d[i]表示机器 i 的运行时间，下标从 0 开始。

count[ ]：数组，长度为 m，其中元素 count[i]表示机器 i 运行的任务数，下标从 0 开始。

i：循环变量。

j：循环变量。

k：临时变量。

max：完成所有任务的时间。

min：临时变量。

(2) 函数 schedule。

```
void  schedule(){
     int i,j,k, max=0;
     for(i=0;i<m;i++){
          d[i]=0;
         for(j=0;j<n;j++){
          s[i][j]=0;
      }
}
for(i=0;i<m;i++){                   //分配前 m 个任务
   s[i][0]=i;
 (1) 
count[i]=1;
}
for( (2) ;i<n;i++){                 //分配后 n-m 个任务
int min=d[0];
k=0;
for(j=1;j<m;j++){                   //确定空闲机器
if(min>d[j]){
   min=d[j];
   k=j;                             //机器 k 空闲
}
}
 (3) ;
count[k]=count[k]+1;
d[k]=d[k]+t[i];

for(i=0;i<m;i++){                   //确定完成所有任务所需要的时间
        if( (4) ){
   max=d[i];
}
     }
     }
}
```

【问题 1】(8 分)

根据说明和 C 代码，填充 C 代码中的空(1)~(4)。

【问题 2】(2 分)

根据说明和 C 代码，该问题采用了 (5) 算法设计策略，时间复杂度为 (6) (用 O

符号表示)。

【问题 3】(5 分)

考虑实例 m=3(编号 0~2)，n=7(编号 0~6)，各任务的运行时间为{16,14,6,5,4,3,2}。则在机器 0、1 和 2 上运行的任务分别为 __(7)__、__(8)__ 和 __(9)__ (给出任务编号)。从任务开始运行到完成所需要的时间为 __(10)__。

解析：

本题考查算法的设计和分析技术中的贪心算法。

贪心算法是一种不追求最优解，只希望得到较为满意解的方法。贪心算法一般可以快速得到满意的解，因为它省去了为找到最优解要穷尽所有可能而必须耗费的大量时间。贪心算法常以当前情况为基础做出最优选择，而不考虑各种可能的整体情况，所以贪心算法不需要回溯。

【问题1】

根据上述思想和题中的说明，首先将s[ ][ ]和d[ ]数组初始化为0，然后将前m个运行时间最长的任务分给m个机器，空(1)处需要表示此时每个机器运行的时间，即当前已经运行的时间加上此时所运行任务的时间，可以推断空(1)处应填入d[i] = d[i] + t[i]。此后需将剩下的n−m个任务按顺序分配给空闲的机器，故空(2)处将i初始化为以m为起始的任务，即空(2)处应填入i=m。空(3)处根据空闲的机器分配任务，所以需记录第k个空闲机器所运行任务的编号，即空(3)处应填入s[k][0]=i。空(4)处已经完成了任务的运行，此处需要统计所有机器所运行任务的最长时间，机器i的运行时间为d[i]，若有d[i]大于当前的最大时间max，就将当前机器的运行时间d[i]赋给max，即空(4)处应填入Max<d[i]。

【问题 2】

根据以上分析，该问题采用了贪心算法的策略，而时间复杂度由算法中的两个嵌套 for 循环和两个非嵌套 for 循环确定，即为 $O(2m\times n+2m)$。

【问题 3】

根据题中算法的思想，将前三个任务分给三个机器，再将接下来的任务分给最先空闲的机器，故可知机器 0 运行任务 0，机器 1 运行任务 1、5，机器 2 运行任务 2、3、4、6，且运行的最长时间为 17。

答案：

【问题1】

(1) d[i] = d[i] + t[i]。(2) i = m。(3) s[k][0] = i。(4) max < d[i]。

【问题 2】

(5) 贪心。(6) $O(2m\times n+2m)$。

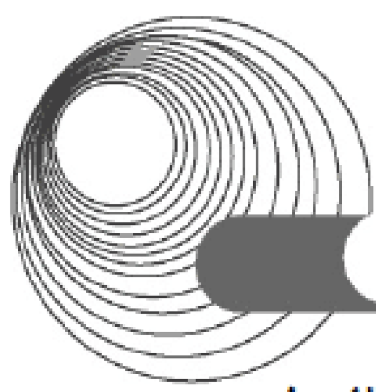
【问题3】

(7) 0。(8) 1、5。(9) 2、3、4、6。(10) 17。

例 2　阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2012 年 11 月试题四)

【说明】

设有 n 个货物要装入若干个容重为 C 的集装箱以便运输，这 n 个货物的体积分别为 $\{s_1, s_2, \cdots, s_n\}$，且有 $s_i \leqslant C(1\leqslant i\leqslant n)$。为节省运输成本，要用尽可能少的集装箱来装运这 n



个货物。

下面分别采用最先适宜策略和最优适宜策略来求解该问题。

最先适宜策略(firstfit)首先将所有的集装箱初始化为空，对于所有货物，按照所给的次序，每次将一个货物装入第一个能容纳它的集装箱中。

最优适宜策略(bestfit)与最先适宜策略类似，不同的是，总是把货物装到能容纳它且目前剩余容重最小的集装箱，使得该箱子装入货物后闲置空间最小。

【C 代码】

下面是这两个算法的 C 语言核心代码。

(1) 变量说明。

n：货物数。

C：集装箱容量。

s：数组，长度为 n，其中每个元素表示货物的体积，下标从 0 开始。

b：数组，长度为 n，b[i]表示第 n+i 个集装箱当前已经装入货物的体积，下标从 0 开始。

i,j：循环变量。

k：所需的集装箱数。

min：当前所用的各集装箱装入了第 i 个货物后的最小剩余容量。

m：当前所需的集装箱数。

temp：临时变量。

(2) 函数 firstfit。

```
int firstfit(){
    int i,j;
    k=0;
    for(i=0;i<n;i++){
        b[i]=0;
    }
    for(i=0;i<n;i++){
        ___(1)___;
        while(C-b[j]<s[i]){
            j++;
        }
        ___(2)___;
        k=k>(j+1)?k:(j+1);
    }
    return k;
}
```

(3) 函数 bestfit。

```
int bestfit(){
    int i,j,min,m,temp;
    k=0;
    for(i=0;i<n;i++){
        b[i]=0;
    }
```

```
    for(i=0;i<n;i++){
        min=C;
        m=k+1;
        for(j=0;j<k+1;j++){
            temp=C-b[j]-s[i];
            if(temp>0 && temp<min){
                ___(3)___;
                m=j;
            }
        }
        ___(4)___;
        k=k>(j+1)?k:(j+1);
    }
    return k;
}
```

【问题 1】(8 分)

根据说明和 C 代码，填充 C 代码中的空(1)~(4)。

【问题 2】(4 分)

根据说明和 C 代码，该问题在最先适宜和最优适宜策略下分别采用了 (5) 和 (6) 算法设计策略，时间复杂度分别为 (7) 和 (8) (用 O 符号表示)。

【问题 3】(3 分)

考虑实例 n=10，C=10，各个货物的体积为{4,2,7,3,5,4,2,3,6,2}。该实例在最先适宜和最优适宜策略下所需的集装箱分别为 (9) 和 (10) 。考虑一般的情况，这两种求解策略能否确保得到最优解？ (11) (能或否)

解析：

本题考查算法的 C 语言实现、时间复杂度的计算及应用，试题本身具有一定的难度。

【问题 1】

本题描述的算法包括最先适宜算法和最优适宜算法。其中，最先适宜算法要求按顺序给货物找到一个能容纳它的集装箱，找到即可装箱。这里的关键在于找到第一个能容纳它的集装箱，从头到尾遍历各集装箱即可。firstfit 函数用于实现最先适宜算法。定位到空(1)处，其上面的 for 循环用于对所有 n 个货物进行遍历，分别找出满足条件 C−b[j]>=s[i]的集装箱。但条件 C−b[j]<s[i]中的变量 j 在空(1)前并没有显式的赋值语句，且遍历各集装箱应从第一个开始，因此空(1)处应填入 j=0。空(2)处表示货物已放入集装箱的情况，应更新装入货物后的体积，因此空(2)处应填入 b[j]= b[j]+s[i]。

最优适宜算法不仅要找到能容纳货物的集装箱，而且还要求该集装箱的剩余容量最小。bestfit 函数用于实现最优适宜算法。该函数的 for 循环语句中的 temp 表示剩余的最小容量，若其小于 min，则应更新其值。因此，空(3)处应填入 min=temp。和 firstfit 函数中空(2)处类似的思路，空(4)处应填入 b[m]= b[m]+s[i]。

【问题 2】

贪心算法在解决最优化问题上是仅根据当前已有的信息做出选择，即不是从整体最优考虑，它所做出的选择只是力求局部最优。最先适宜策略和最优适宜策略均采用了该算法设计策略。

对于时间复杂度，应根据程序中循环的层数及每层循环的次数来进行计算。可以很容易地判断，这两种算法的时间复杂度均为 $O(n^2)$。

【问题 3】

本问题考查对程序的具体理解和应用。

对 firstfit 函数进行遍历的结果如表 5-1 所示。

表 5-1　对 firstfit 函数进行遍历的结果

i	b[j]	k
0	b[0]=4	1
1	b[0]=6	1
2	b[1]=7	2
3	b[1]=10	2
4	b[2]=5	3
5	b[2]=9	3
6	b[3]=2	4
7	b[3]=5	4
8	b[4]=6	5
9	b[4]=8	5

因此，该实例在最先适宜策略下所需的集装箱数为 5。同理可对 bestfit 函数进行遍历，可得到该实例在最优适宜策略下所需的集装箱数为 4，遍历过程可由考生自己进行，以增强对整个算法的理解。

由于贪心算法在解决最优化问题上是仅根据当前已有的信息做出选择，即不是从整体最优考虑，它所做出的选择只是力求局部最优，因此这两种求解策略均不能确保得到最优解。

答案：

【问题 1】

(1) j=0。　(2) b[j]= b[j]+s[i]。　(3) min=temp。　(4) b[m]= b[m]+s[i]。

【问题 2】

(5) 贪心。　(6) 贪心。　(7) $O(n^2)$。　(8) $O(n^2)$。

【问题 3】

(9) 5。　(10) 4。　(11) 否。

例 3　阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2012 年 5 月试题四)

【说明】

用两台处理机 A 和 B 处理 n 个作业。设 A 和 B 处理第 i 个作业的时间分别为 a_i 和 b_i。由于各个作业的特点和机器性能的关系，对某些作业，在 A 上处理时间长，而对某些作业，在 B 上处理时间长。一台处理机在某个时刻只能处理一个作业，而且作业处理是不可中断的，每个作业只能被处理一次。现要找出一个最优调度方案，使得 n 个作业被这两台处理机处理完毕的时间(所有作业被处理的时间之和)最少。

算法步骤：

(1) 确定候选解上界为 R 短的单台处理机处理所有作业的完成时间 m，有

$$m = \min(\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} b_i)$$

(2) 用 p(x, y, k)=1 表示前 k 个作业可以在 A 用时不超过 x 且在 B 用时不超过 y 时间内处理完成，则 $p(x,y,k)=p(x-a_k, y, k-1)||p(x, y-b_k, k-1)$(||表示逻辑或操作)。

(3) 得到最短处理时间为 min(max(x,y))。

【C 代码】

下面是该算法的 C 语言实现。

(1) 常量和变量说明。

n：作业数。

m：候选解上界。

a：数组，长度为 n，记录 n 个作业在 A 上的处理时间，下标从 0 开始。

b：数组，长度为 n，记录 n 个作业在 B 上的处理时间，下标从 0 开始。

k：循环变量。

p：三维数组，长度为(m+1)*(m+1)*(n+1)。

temp：临时变量。

max：最短处理时间。

(2) C 代码。

```
#include <stdio.h>
int n, m;
int a[60], b[60], p[100][100][60];
void read(){ /*输入n、a、b，求出m，代码略*/}
void schedule(){ /*求解过程*/
  int x,y,k;
  for(x=0;x<=m;x++){
    for(y=0;y<m;y++){
      ___(1)___
      for(k=1;k<n;k++)
         p[x][y][k]=0;
    }
  }
  for(k=1;k<n;k++){
    for(x=0;x<=m;x++){
      for(y=0;y<=m;y++){
        if(x - a[k-1]>=0) ___(2)___;
        if(___(3)___)p[x][y][k]=(p[x][y][k] ||p[x][y-b[k-1]][k-1]);
      }
    }
```

```
    }
  }
  void write(){  /*确定最优解并输出*/
    int x,y,temp,max=m;
    for(x=0;x<=m;x++){
      for(y=0;y<=m;y++){
        if(__(4)__){
          temp=__(5)__;
          if(temp<max)max = temp;
        }
      }
    }
    printf("\n%d\n",max);
  }
  void main(){read();schedule();write();}
```

【问题 1】(9 分)

根据以上说明和 C 代码，填充 C 代码中的空(1)~(5)。

【问题 2】(2 分)

根据以上 C 代码，算法的时间复杂度为__(6)__(用 O 符号表示)。

【问题 3】(4 分)

考虑 6 个作业的实例，各个作业在两台处理机上的处理时间如表 5-2 所示。该实例的最优解为__(7)__，最优解的值(即最短处理时间)为__(8)__。最优解用(x_1, x_2, x_3, x_4, x_5, x_6)表示，其中若第 i 个作业在 A 上处理，则 x_i=l，否则 x_i=2。如(1, 1, 1, 1, 2, 2)表示作业 1, 2, 3 和 4 在 A 上处理，作业 5 和 6 在 B 上处理。

表 5-2　各作业在两台处理机上的处理时间

	作业 1	作业 2	作业 3	作业 4	作业 5	作业 6
处理机 A	2	5	7	10	5	2
处理机 B	3	8	4	11	3	4

解析：

本题考查独立任务最优调度问题，也称双机调度，是一种动态规划算法。

【问题 1】

从 schedule()函数的第一个程序段可以看出，该段程序主要进行初始化第一个作业，下标以 0 开始，即 p[x][y][0]=1，内层循环里的 p[x][y][k]=0 用于初始化后面的 n−1 个作业。第二个程序段是对后面的 n−1 个作业，确定 p(x, y, k)的值。x−a[k−1]>=0 的判定条件若成立，则表示第 k 个作业由机器 A 处理，完成 k−1 个作业时机器 A 花费的时间是 x−a[k−1]，即 p[x][y][k]=p[x-a[k−1]][y][k−1]。空(3)处要求填入一判定条件，由其后的执行语句可知，第 k 个作业由机器 B 处理，因此判定条件应为 y−b[k−1]>=0。

write()程序段用于确定最优解并输出结果，即得到最短处理时间 min(max(x,y))。空(4)处的判定条件是任务 n 完成，因此为 p[x][y][n]==1 或其等价形式。空(5)处表达 max(x,y)，为(x>=y)?x:y。

【问题 2】

从程序的循环层数即可看出算法的时间复杂度。程序的最高循环层数为 3 层。最外层循环变量的变化范围是 1~n−1，次外层循环变量的变化范围是 0~m，内层循环变量的变化范围是 0~m，所以时间复杂度为 $O(m^2n)$。

【问题 3】

为了方便考生更好地理解本算法的思想，现做如下分析。

当完成 k 个作业，设机器 A 花费了 x 时间，机器 B 所花费时间的最小值肯定是 x 的一个函数。设 F[k][x]表示机器 B 所花费时间的最小值，则 F[k][x]=min{F[k−1][x]+b[k], F[k−1][x−a[k]]}。其中，F[k−1][x]+b[k]表示第 k 个作业由机器 B 来处理(完成 k−1 个作业时机器 A 花费的时间仍是 x)，F[k−1][x−a[k]]表示第 k 个作业由机器 A 处理(完成 k−1 个作业时机器 A 花费的时间是 x−a[k])。

那么单个点对较大值 max(x, F[k][x])即表示此时(即机器 A 花费 x 时间的情况下)所需要的总时间。而机器 A 花费的时间 x 是变化的，即 x=0,1,2,…,x(max)，由此构成了点对较大值序列。要求整体时间最短，取这些点对较大值序列中最小的即可。现分析前两个作业的情况。

对于第一个作业：下标以 0 开始。

机器 A 所花费时间的所有可能值范围是 0 <= x <= a[0]。

x<0 时，设 F[0][x]=∞，则 max(x,∞)=∞；记法意义见下。

x=0 时，F[0][0]=3，则 max(0,3)=3，机器 A 花费 0 时间，机器 B 花费 3 时间，而此时两个机器所需时间为 3。

x=1 时，F[0][1]=3，max(1,3)=3。

x=2 时，F[0][2]=0，则 max(2,0)=2。

在上面的点对序列中可以看出，当 x=2 时，完成第一个作业两台机器花费最少的时间为 2，此时机器 A 花费 2 时间，机器 B 花费 0 时间。

再来看第二个作业。

x 的取值范围是 0 <= x <= (a[0] + a[1])。

当 x<0 时，记 F[1][x] =∞；这个记法编程使用，因为数组下标不能小于 0。在这里的实际含义是：x 是代表完成前两个作业机器 A 的时间，a[1]是机器 A 完成第二个作业的时间，若 x<a[1]，则势必第二个作业由机器 B 来处理，即在 min()中取前者。

x=0，则 F[1][0]= min{ F[0][0]+b[2], F[0][0−a[1]] }= min{3+8,∞}=11，进而 max(0,11)=11。

x=1，则 F[1][1]= min{ F[0][1]+b[2], F[0][1−a[1]] }= min{3+8,∞}=11，进而 max(1,11)=11。

x=2，则 F[1][2]= min{ F[0][2]+b[2], F[0][2−a[1]] }= min{0+8,∞}=8，进而 max(2,8)=8。

x=3，则 F[1][3]= min{ F[0][3]+b[2], F[0][3−a[1]] }= min{0+8,∞}=8，进而 max(3,8)=8。

x=4，则 F[1][4]= min{ F[0][4]+b[2], F[0][4−a[1]] }= min{0+8,∞}=8，进而 max(4,8)=8。

x=5，则 F[1][5]= min{ F[0][5]+b[2], F[0][5−a[1]] }= min{0+8,3}=3，进而 max(5,3)=5。

x=6，则 F[1][6]= min{ F[0][6]+b[2], F[0][6−a[1]] }= min{0+8,3}=3，进而 max(6,3)=6。

x=7，则 F[1][7]= min{ F[0][7]+b[2], F[0][7−a[1]] }= min{0+8,0}=0，进而 max(7,0)=7。

在上面的点对序列中可以看出，当 x=5 时，完成两个作业两台机器花费最少的时间为 5，此时机器 A 花费 5 时间，机器 B 花费 3 时间。

接下来依次类推即可，最终该实例的最优解为(1,1,2,2,1,1)，最短处理时间为 15。这里提供当各个作业完成时的最短处理时间，考生可自行推导：2，5，7，12，14，15。

答案：

【问题 1】

(1) p[x][y][0]=1。 (2) p[x][y][k]= p[x−a[k−1]][y][k−1]。 (3) y−b[k−1]>=0。

(4) p[x][y][n]==1 或 p[x][y][n] 或 p[x][y][n]!=0。 (5) (x>=y) ?x:y。

【问题 2】

(6) $O(m^2n)$。

【问题 3】

(7) (1,1,2,2,1,1)。(8) 15。

例 4 阅读下列说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2011 年 11 月试题四)

【说明】(共 15 分)

设某一机器由 n 个部件组成，每一个部件都可以从 m 个不同的供应商处购得。供应商 j 供应的部件 i 具有重量 W_{ij} 和价格 C_{ij}。设计一个算法，求解总价格不超过上限 cc 的最小重量的机器组成。

采用回溯法来求解该问题。

首先定义解空间。解空间由长度为 n 的向量组成，其中每个分量取值来自集合{l, 2, …, m}，将解空间用树型结构表示。

接着从根节点开始，以深度优先的方式搜索整个解空间。从根节点开始，根节点成为活节点，同时也成为当前的扩展节点。向纵深方向考虑第一个部件从第一个供应商处购买，得到一个新节点。判断当前的机器价格(C_{11})是否超过上限(cc)，重量(W_{11})是否比当前已知的解(最小重量)大，若是，应回溯至最近的一个活节点；否则，该新节点成为活节点，同时也成为当前的扩展节点，根节点不再是扩展节点。继续向纵深方向考虑第二个部件从第一个供应商处购买，得到一个新节点。同样判断当前的机器价格($C_{11}+C_{21}$)是否超过上限(cc)，重量($W_{11}+W_{21}$)是否比当前已知的解(最小重量)大。若是，应回溯至最近的一个活节点；否则，该新节点成为活节点，同时也成为当前的扩展节点，原来的节点不再是扩展节点。以这种方式递归地在解空间中搜索，直到找到所要求的解或者解空间中已无活节点为止。

【C 代码】

下面是该算法的 C 语言实现。

(1) 变量说明。

n：机器的部件数。

m：供应商数。

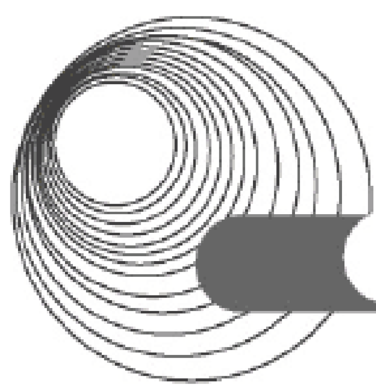
cc：价格上限。

w[][]：二维数组，w[i][j]表示第 j 个供应商供应的第 i 个部件的重量。

c[][]：二维数组，c[i][j]表示第 j 个供应商供应的第 i 个部件的价格。

bestW：满足价格上限约束条件的最小机器重量。
bestC：最小重量机器的价格。
bestX[]：最优解，一维数组，bestX[i]表示第 i 个部件来自哪个供应商。
cw：搜索过程中机器的重量。
cp：搜索过程中机器的价格。
x[]：搜索过程中产生的解，x[i]表示第 i 个部件来自哪个供应商。
i：当前考虑的部件，从 0 到 n-1。
j：循环变量。
(2) 函数 backtrack 的实现如下。

```
int n=3;
int m=3;
int cc=4;
int w[3][3]={{1,2,3},{3,2,1},{2,2,2}};
int c[3][3]={{1,2,3},{3,2,1},{2,2,2}};
int bestW=8;
int bestC=0;
int bestX[3]={0,0,0};
int cw=0;
int cp=0;
int x[3]={0,0,0};
int backtrack(int i){
   int j=0;
   int found=0;
   if(i>n-1){/*得到问题解*/
       bestW=cw;
       bestC=cp;
       for(j=0;j<n;j++){
      __(1)__;
     }
       return 1;
  }
  if(cp<=cc){/*有解*/
       found=1;
  }
  for(j=0; __(2)__;j++){
     /*第 i 个部件从第 j 个供应商购买*/
    __(3)__;
       cw=cw+w[i][j];
       cp=cp+c[i][j];
       if(cp<=cc && __(4)__ {/*深度搜索，扩展当前节点*/
           if(backtrack(i+1)){found=1;}
       }
       /*回溯*/
       cw=cw-w[i][j];
    __(5)__;
  }
```




```
        return found;
    }
```

解析：

设本题中机器需要 3 个部件，共 3 个供应商，每个供应商可提供 3 种部件，供应商 0 提供的 3 个部件数量分别为 1、2、3，价格分别为 1、2、3；供应商 1 提供的 3 个部件数量分别为 3、2、1，价格分别为 3、2、1；供应商 2 提供的 3 个部件数量分别为 2、2、2，价格分别为 2、2、2。

价格上限为 4；初始时，满足价格上限约束条件的最小机器重量为 8，最小重量机器的价格为 0。

在回溯过程中，先购买第 0 个部件，首先选择第 0 个供应商的部件 0，计算总重量和总价格，如果总价格不大于上限 cc，则扩展当前节点；然后购买第 1 个部件，同样先选择第 0 个供应商的部件 1，计算总重量和总价格，如果总价格不大于上限 cc，则扩展当前节点，如果当前总价格大于上限 cc 或者当前总重量比已知的最小重量大，则当前节点成为死节点，返回前一次购买部件所在的节点，同时更新总价格和总重量。因此可将空(2)~(5)补充完整，如下。

```
for(j=0;  j<m; j++){
   /*第 i 个部件从第 j 个供应商购买*/
     x[i]=j;
     cw=cw+w[i][j];
     cp=cp+c[i][j];
     if(cp<=cc &&  cw<bestW {/*深度搜索，扩展当前节点*/
         if(backtrack(i+1)){found=1;}
     }
     /*回溯*/
     cw= cw -w[i][j];
     cp= cp - c[i][j];
}
```

如果得到问题解，将部件的总重量和总价格保存在变量 bestW 和 bestC 中，并将部件的来源保存在数组 bestX 中。数组 x 中保存搜索过程中产生的解，把 x 中的元素值赋给数组 bestX 即可。因此，空(1)处应填入 bestX[j]=x[j]。

答案：

(1) bestX[j]=x[j]。　　(2) j<m。　　(3) x[i]=j

(4) cw< bestW。　　(5) cp= cp - c[i][j]

例 5　阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2011 年 5 月试题四)

【说明】

某应用中需要对 100 000 个整数元素进行排序，每个元素的取值在 0~5 之间。排序算法的基本思想是：对每一个元素 x，确定小于等于 x 的元素个数(记为 m)，将 x 放在输出元素序列的第 m 个位置。对于元素值重复的情况，依次放入第 m−1, m−2, …个位置。例如，如果元素值小于等于 4 的元素个数有 10 个，其中元素值等于 4 的元素个数有 3 个，则 4 应该在数据元素序列的第 10 个位置、第 9 个位置和第 8 个位置上。

算法具体的步骤如下。

步骤 1：统计每个元素值的个数。

步骤 2：统计小于等于每个元素值的个数。

步骤 3：将输入元素序列中的每个元素放入有序的输出元素序列。

【C 代码】

下面是该排序算法的 C 语言实现。

(1) 常量和变量说明。

R：常量，定义元素取值范围中的取值个数，如上述应用中 R 值应取 6。

i：循环变量。

n：待排序元素个数。

a：输入数组，长度为 n。

b：输出数组，长度为 n。

c：辅助数组，长度为 R，其中每个元素表示小于等于下标所对应的元素值的个数。

(2) 函数 sort。

```
1   void sort(int n, int a[ ], int b[ ]){
2       int c[R], i;
3       for(i=0; i<___(1)___; i++){
4           c[i]=0;
5       }
6       for(i=0; i<n; i++){
7           c[a[i]]=___(2)___;
8       }
9       for(i=0; i<R; i++){
10          c[i]=___(3)___;
11      }
12      for(i=0; i<n; i++){
13          b[c[a[i]]-1]=___(4)___;
14          c[a[i]]=c[a[i]]-1;
15      }
16  }
```

【问题 1】(8 分)

根据说明和 C 代码，填充 C 代码中的空缺(1)~(4)。

【问题 2】(4 分)

根据 C 代码，函数的时间复杂度和空间复杂度分别为___(5)___和___(6)___(用 O 符号表示)。

【问题 3】(3 分)

根据以上 C 代码，分析该排序算法是否稳定。若稳定，请简要说明(不超过 100 字)；若不稳定，请修改其中代码使其稳定(给出要修改的行号和修改后的代码)。

解析：

【问题 1】

在函数 sort 中，首先对辅助数组 c 进行初始化，将 R 个元素全部初始化为 0，代码为：

```
for(i=0; i<R; i++){
    c[i]=0;
}
```

然后，统计每个元素值的个数，并将元素的个数保存在数组 c 中，c 的下标表示对应的元素。通过循环读取元素 a[i]，每读取到一个 a[i]，c[a[i]]的值便加 1。代码为：

```
for(i=0; i<n; i++){
    c[a[i]]= c[a[i]]+1;
}
```

接下来统计小于等于每个元素值的个数，用 c[i]保存。输入数组的每个元素的取值在 0~5 之间，即有 6 种取值(0、1、2、3、4、5)，则 c[0]保存的是值为 0 的元素个数，c[1]保存的是值为 1 的元素个数，依次类推，c[5]保存的是值为 5 的元素个数。则小于等于 0 的元素有 c[0]个，小于等于 1 的元素有 c[0]+ c[1]个，小于等于 2 的元素有 c[0]+ c[1] +c[2]个，依次类推，小于等于 5 的元素个数为 c[0]+ c[1]+ c[2]+ c[3]+c[4]+c[5]。代码为：

```
for(i=0; i<R; i++){
    c[i]= c[i]+c[i-1];
}
```

最后，将输入元素序列中的每个元素放入有序的输出元素序列。小于等于 a[i]的元素有 c[a[i]]个，由于 C 语言中，数组的下标从 0 开始，每读取的一个值为 a[i]的元素应保存到 b[c[a[i]]−1]，然后将 c[a[i]]的值减 1，c[a[i]]保存当前未保存到数组 b 中的值为 a[i]的元素个数。代码为：

```
for(i=0; i<n; i++){
    b[c[a[i]]-1]= a[i];
    c[a[i]]=c[a[i]]-1;
}
```

【问题 2】

算法中只有单层循环，因此算法的时间复杂度为 O(n+R)。

算法中需要用到一个长度为 R 的辅助数组 c，因此算法的空间复杂度为 O(R)。

【问题 3】

假定在待排序的记录序列中存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中，r_i=r_j，且 r_i 在 r_j 之前，而在排序后的序列中，r_i 仍在 r_j 之前，则称这种排序算法是稳定的；否则称为不稳定的。题目中，从下标 0 开始读取 a[i]元素，第一个读取的值为 a[i]的元素保存在 b[c[a[i]]−1]，第二个读取的 a[i]的元素保存在 b[c[a[i]]−2]，也就是说后读取的值为 a[i]的元素保存在前面，因此该算法是不稳定的，只需将最后一个 for 循环改为如下代码即可变为稳定的。

```
for(i=n-1;i>=0;i--){
    b[c[a[i]]-1]= a[i];
    c[a[i]]=c[a[i]]-1;
}
```

答案：

【问题 1】

(1) R。　(2) c[a[i]]+1。　(3) c[i]+c[i−1]。　(4) a[i]。

【问题 2】

(5) O(n+R)或 O(n)或 n 或线性。

(6) O(R)或 O(1)。

【问题 3】

不稳定。修改第 12 行的 for 循环为“for(i=n−1;i>=0;i−−){”即可。

例 6　阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2010 年 11 月试题四)

【说明】

堆数据结构定义如下。

对于 n 个元素的关键字序列｛$a_1, a_2, \cdots, a_n$｝，当且仅当满足下列关系时称其为堆：

$$\begin{cases} a_i \leqslant a_{2i} \\ a_i \leqslant a_{2i+1} \end{cases} \text{或} \begin{cases} a_i \geqslant a_{2i} \\ a_i \geqslant a_{2i+1} \end{cases} \quad \text{其中，} i=1,2,\cdots,\left\lfloor \frac{n}{2} \right\rfloor$$

在一个堆中，若堆顶元素为最大元素，则称为大顶堆；若堆顶元素为最小元素，则称为小顶堆。堆常用完全二叉树表示，图 5-1 是一个大顶堆的例子。

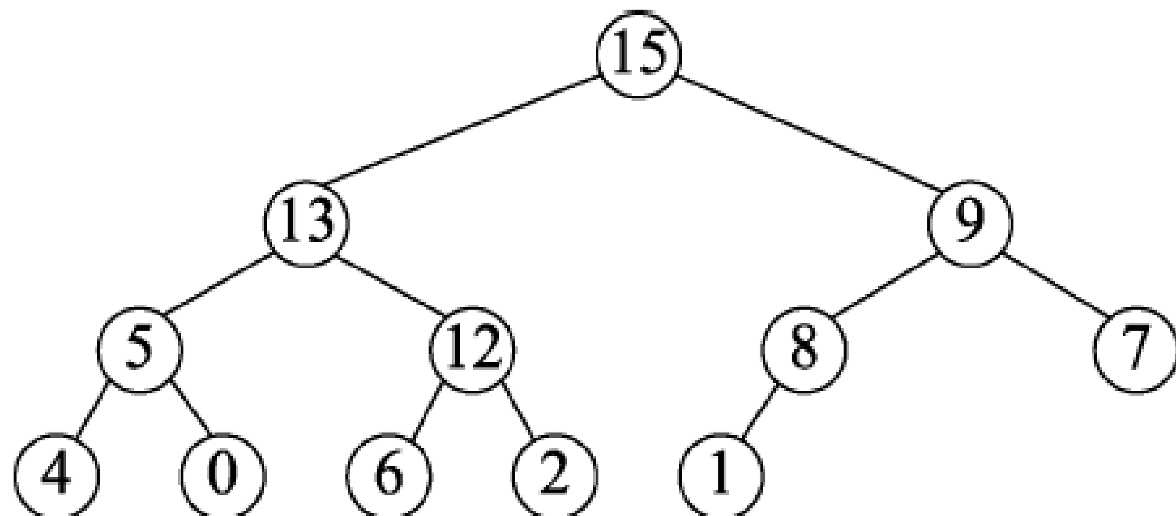


图 5-1　大顶堆示例

堆数据结构常用于优先队列中，以维护由一组元素构成的集合。对应于两类堆结构，优先队列也有最大优先队列和最小优先队列，其中最大优先队列采用大顶堆，最小优先队列采用小顶堆。以下考虑最大优先队列。

假设现已建好大顶堆 A，且已经实现了调整堆的函数 heapify(A,n,index)。

下面将 C 代码中需要完善的三个函数说明如下。

(1) heapMaximum(A)：返回大顶堆 A 中的最大元素。

(2) heapExtractMax(A)：去掉并返回大顶堆 A 的最大元素，将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆。

(3) maxHeapInsert(A, key)：把元素 key 插入到大顶堆 A 的最后位置，再将 A 调整成大顶堆。

优先队列采用顺序存储方式，其存储结构定义如下：

```
#define PARENT(i) i/2
typedef struct array{
    int *int_array;  //优先队列的存储空间首地址
    int array_size;  //优先队列的长度
    int capacity;    //优先队列存储空间的容量
}ARRAY;
```

【C 代码】

(1) 函数 heapMaximum。

```
int heapMaximum(ARRAY *A){return ___(1)___;}
```

(2) 函数 heapExtractMax。

```
int heapExtractMax(ARRAY *A){
    int max;
    max=A->int_array[0];
    ___(2)___;
    A->array_size--;
    Heapify(A,A->array_size,0);//将剩余元素调整成大顶堆
    return max;
 }
```

(3) 函数 maxHeapInsert。

```
int maxHeapInsert(ARRAY *A,int key){
    int i,*p;
    if (A->array-size==A->capacity){//存储空间的容量不够时扩充空间
        p=(int*)realloc(A->int array, A->capacity *2*sizeof(int));
        if(!p) return-1;
        A->int_array=P;
        A->capacity=2*A->capacity;
    }
    A->array_size++:
    i=___(3)___;
    while(i>0 && ___(4)___){
        A->int_array[i]=A->int_array[PARENT(i)];
        i=PARENT(i);
    }
    ___(5)___;
    return 0;
  }
```

【问题 1】(10 分)

根据以上说明和 C 代码，填充 C 代码中的空(1)~(5)。

【问题 2】(3 分)

根据以上 C 代码，函数 heapMaximum、heapExtractMax 和 maxHeapInsert 的时间复杂度的紧致上界分别为__(6)__、__(7)__和__(8)__(用 O 符号表示)。

【问题 3】(2 分)

若将元素 10 插入到堆 A=(15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1)中，调用 maxHeapInsert 函数进行操作，则新插入的元素在堆 A 中第__(9)__个位置(从 1 开始)。

解析：

【问题 1】

heapMaximum(A)函数返回大顶堆 A 中的最大元素。大顶堆 A 的优先队列采用顺序存储方式，指针 int_array 指向优先队列的存储空间首地址，其内容为大顶堆 A 中的最大元素，因此空(1)处应填入 A->int_array[0]。

heapExtractMax(A)的功能是去掉并返回大顶堆 A 的最大元素，将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆。可知空(2)处所填的语句应该是将最后一个元素的值存储在原最大元素所在的位置，即存储空间的首地址。

maxHeapInsert(A, key)的功能是把元素 key 插入到大顶堆 A 的最后位置，再将 A 调整成大顶堆。在将 A 调整成大顶堆的过程中需要用到上滤策略。maxHeapInsert(A, key)函数中，首先用 i 指示元素 key 的位置，则 i= array_size−1；然后将 int_array[i]与其父节点进行比较，如果大于其父节点的值，将两者的位置进行交换，key 的位置 i=PARENT(i)；往上比较，直至 key 的值不大于其父节点的值。

【问题 2】

heapMaximum(A)函数不需要进行比较，直接输出存储空间首地址中的内容。时间复杂度的紧致上界为 O(1)。

heapExtractMax(A)函数将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆，在最坏的情况下，需要从根节点下滤比较到最底层，时间复杂度的紧致上界为 O(lgn)。

maxHeapInsert(A, key)函数把元素 key 插入到大顶堆 A 的最后位置，再将 A 调整成大顶堆。在最坏的情况下，需要从最底层上滤比较到根节点，时间复杂度的紧致上界为 O(lgn)。

【问题 3】

调用 maxHeapInsert 函数进行排序的过程如图 5-2 所示。

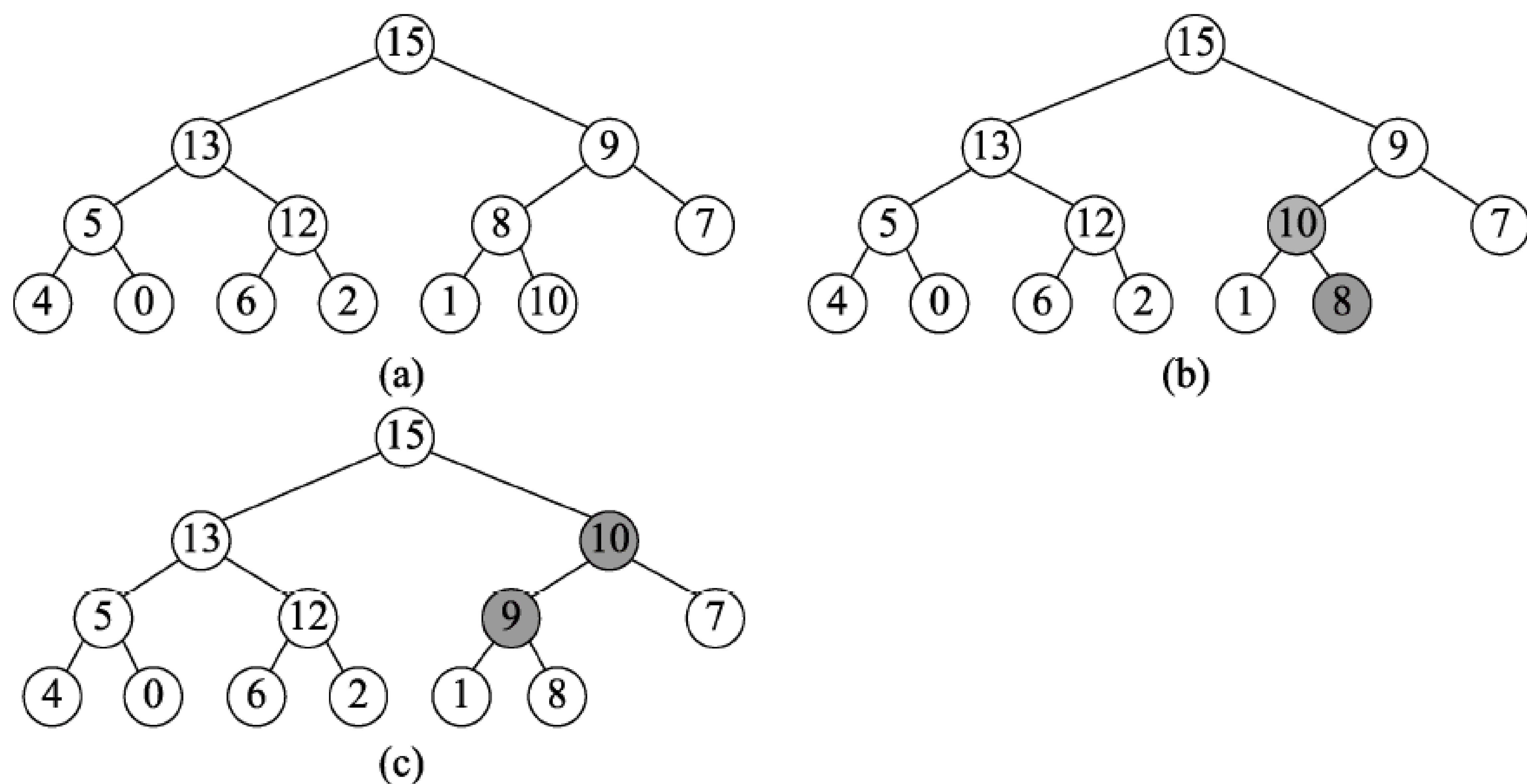


图 5-2　调用 maxHeapInsert 函数进行排序的过程

可见，元素 10 在堆 A 的第 3 个位置。

答案：

【问题 1】

(1) A−>int_array[0]。

(2) A−>int_array[0]=A−>int_array[A−>array_size−1]。

(3) A−>array_size−1。

(4) key>A−>int_array[PARENT(i)]。

(5) A−>int_array[i]=key。

【问题 2】

(6) O(1)。 (7) O(lgn)。 (8) O(lgn)。

【问题 3】

(9) 3。

例 7 阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2010 年 5 月试题四)

【说明】

对有向图进行拓扑排序的方法如下。

(1) 初始时拓扑序列为空。

(2) 任意选择一个入度为 0 的顶点，将其放入拓扑序列中，同时从图中删除该顶点以及从该顶点出发的弧。

(3) 重复步骤(2)，直到不存在入度为 0 的顶点为止(若所有顶点都进入拓扑序列则完成拓扑排序，否则由于有向图中存在回路无法完成拓扑排序)。

函数 int *TopSort(LinkedDigraph G)的功能是对有向图 G 中的顶点进行拓扑排序，返回拓扑序列中的顶点编号序列，若不能完成拓扑排序，则返回空指针。其中，图 G 中的顶点从 1 开始依次编号，顶点序列为 v1, v2, …, vn。图 G 采用邻接表示，其数据类型定义如下：

```
#define MAXVNUM 50                              /*最大顶点数*/
typedef struct ArcNode{                         /*表节点类型*/
 int adjvex;                                    /*邻接顶点编号*/
 struct ArcNode *nextarc;                       /*指示下一个邻接顶点*/
}ArcNode;
typedef struct AdjList{                         /*头节点类型*/
 char vdata;                                    /*顶点的数据信息*/
 ArcNode *firstarc;                             /*指向邻接表的第一个表节点*/
}AdjList;
typedef struct LinkedDigraph{                   /*图的类型*/
 int n;                                         /*图中顶点个数*/
 AdjList Vhead[MAXVNUM];                        /*所有顶点的头节点数组*/
}LinkedDigraph;
```

例如，某有向图 G 如图 5-3 所示，其邻接表如图 5-4 所示。

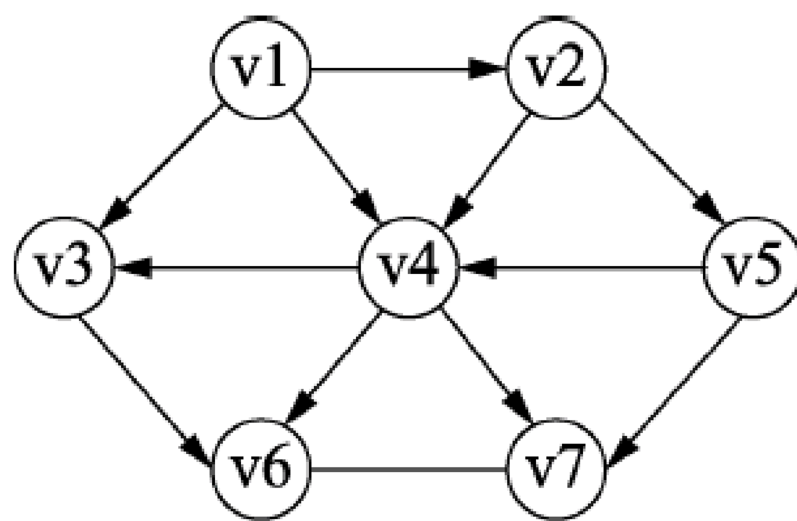


图 5-3 有向图 G

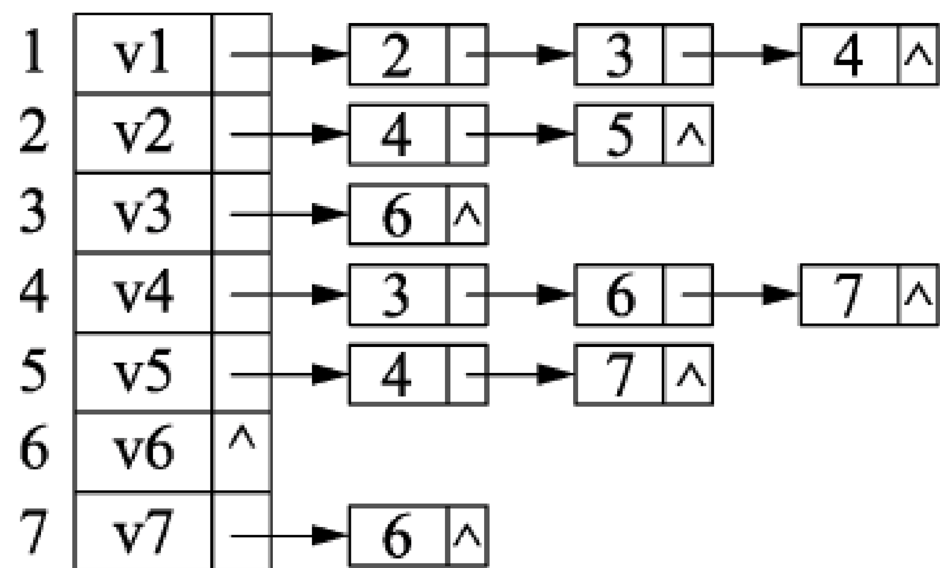


图 5-4 有向图 G 的邻接表示意图

函数 TopSort 中用到了队列结构(Queue 的定义省略)，实现队列基本操作的函数原型如表 5-3 所示。

表 5-3　实现队列基本操作的函数原型

函数原型	说　明
void InitQueue(Queue*Q)	初始化队列(构造一个空队列)
bool IsEmpty(Queue Q)	判断队列是否为空，若是则返回 true，否则返回 false
void EnQueue(Queue*Q,int e)	元素入队列
void DeQueue(Queue*Q,int*p)	元素出队列

【C 代码】

```
int *TopSort(LinkedDigraph G) {
 ArcNode *p;              /*临时指针，指示表节点*/
 Queue Q;                 /*临时队列，保存入度为 0 的顶点编号*/
 int k = 0;               /*临时变量，用作数组元素的下标*/
 int j = 0, w = 0;        /*临时变量，用作顶点编号*/
 int *topOrder, *inDegree;
 topOrder = (int *)malloc((G.n+1) * sizeof(int));  /*存储拓扑序列中的顶点编号*/
 inDegree = (int *)malloc((G.n+1) * sizeof(int));  /*存储图G中各顶点的入度*/
 if (!inDegree || !topOrder) return NULL;
___(1)___;                                         /*构造一个空队列*/
 for ( j = 1; j <= G.n; j++ ) {                    /*初始化*/
   topOrder[j] = 0;      inDegree[j] = 0;
}
for (j = 1; j <= G.n; j++)                         /*求图 G 中各顶点的入度*/
   for( p = G.Vhead[j].firstarc; p; p = p->nextarc )
   inDegree[p-> adjvex] += 1;
for (j = 1; j <= G.n; j++)                /*将图 G 中入度为 0 的顶点保存在队列中*/
 if ( 0 == inDegree[j] ) EnQueue(&Q,j);
while (!IsEmpty(Q)) {
  ___(2)___;                              /*队头顶点出队列并用 w 保存该顶点的编号*/
topOrder[k++] = w;
/*将顶点 w 的所有邻接顶点的入度减 1(模拟删除顶点 w 及从该顶点出发的弧的操作)*/
  for(p = G.Vhead[w].firstarc; p; p = p->nextarc) {
 ___(3)___ -= 1;
  if (0 ==___(4)___) EnQueue(&Q, p->adjvex);
  }/* for */
  }/* while */
  free(inDegree);
  if (___(5)___)
  return NULL;
  return topOrder;
} /*TopSort*/
```

【问题 1】(9 分)

根据以上说明和 C 代码，填充 C 代码中的空(1)~(5)。

【问题 2】(2 分)

对于图 5-3 所示的有向图 G，写出函数 TopSort 执行后得到的拓扑序列。若将函数

TopSort 中的队列改为栈，写出函数 TopSort 执行后得到的拓扑序列。

【问题 3】(4 分)

设某有向无环图的顶点个数为 n、弧数为 e，那么用邻接表存储该图时，实现上述拓扑排序算法的函数 TopSort 的时间复杂度是___(6)___。

若有向图采用邻接矩阵表示(例如，图 5-3 所示有向图的邻接矩阵如图 5-5 所示)，且将函数 TopSort 中有关邻接表的操作修改为针对邻接矩阵的操作，那么对于有 n 个顶点、e 条弧的有向无环图，实现上述拓扑排序算法的时间复杂度是___(7)___。

	v1	v2	v3	v4	v5	v6	v7
v1	0	1	1	1	0	0	0
v2	0	0	0	1	1	0	0
v3	0	0	0	0	0	1	0
v4	0	0	1	0	0	1	1
v5	0	0	0	1	0	0	1
v6	0	0	0	0	0	0	0
v7	0	0	0	0	0	1	0

图 5-5　有向图 G 的邻接矩阵

解析：

本题考查数据结构和算法中的拓扑排序算法。

【问题 1】

在拓扑排序过程中，需要将入度为 0 的顶点临时存储起来。函数中用一个队列暂存入度为 0 且没有进入拓扑序列的顶点。显然，空(1)处应填入 InitQueue(&Q)。

根据注释，空(2)处应填入 DeQueue(&Q, &w)，实现队头元素出队列的处理。

题中图采用邻接表存储结构，当指针 p 指向 vi 邻接表中的节点时，p->adjvex 表示 vi 的一个邻接顶点，删除 vi 至顶点 p->adjvex 的弧的操作实现为顶点 p->adjvex 的入度减 1，因此，空(3)处应填入 inDegree[p->adjvex]。当顶点 p->adjvex 的入度为 0 时，需要将其加入队列，因此空(4)处也应填入 inDegree[p->adjvex]。

空(5)处判断是否所有顶点都加入拓扑序列，算法中变量 k 用于对加入序列的顶点计数，因此，空(5)处应填入 k<G.n 或 k!=G.n。

【问题 2】

使用栈和队列的差别在于拓扑序列中顶点的排列次序可能不同。对于本题中的有向图，在使用队列的方式下：

① 开始时仅顶点 v1 的入度为 0，因此顶点 v1 入队。

② 队头顶点 v1 出队，并进入拓扑序列，然后删除从顶点 v1 出发的弧后，仅使顶点 v2 的入度为 0，因此顶点 v2 入队。

③ 队头顶点 v2 出队，并进入拓扑序列，然后删除从顶点 v2 出发的弧后，仅使顶点 v5 的入度为 0，因此顶点 v5 入队。

④ 队头顶点 v5 出队，并进入拓扑序列，然后删除从顶点 v5 出发的弧后，仅使顶点 v4 的入度为 0，因此顶点 v4 入队。

⑤ 队头顶点 v4 出队，并进入拓扑序列，然后删除从顶点 v4 出发的弧后，仅使顶点 v3 和 v7 的入度为 0，因此顶点 v3 和 v7 依次入队。

⑥ 队头顶点 v3 出队，并进入拓扑序列，然后删除从顶点 v3 出发的弧后，没有产生新的入度为 0 的顶点。

⑦ 队头顶点 v7 出队，并进入拓扑序列，然后删除从顶点 v7 出发的弧后，使顶点 v6 的入度为 0，因此顶点 v6 入队。

⑧ 队头顶点 v6 出队，并进入拓扑序列，然后删除从顶点 v6 出发的弧后，没有产生新的入度为 0 的顶点，队列已空，因此结束拓扑排序过程，得到的拓扑序列为 v1 v2 v5 v4 v3 v7 v6。

使用栈保存入度为 0 的顶点时，前 4 步都是一样的，因为每次仅有一个元素进栈，因此出栈序列与入栈序列一致。到第⑤步时，v3 和 v7 依次入栈后，出栈时的次序为 v7 和 v3，因此得到的拓扑序列为 v1 v2 v5 v4 v7 v3 v6。

【问题 3】

以邻接表为存储结构时，计算各顶点入度的时间复杂度为 O(e)，建立 0 入度顶点队列的时间复杂度为 O(n)。在拓扑排序过程中，(图中无环情况下)每个顶点进出队列各 1 次，入度减 1 的操作在 while 循环中共执行 e 次，所以总的时间复杂度为 O(n+e)。

以邻接矩阵为存储结构时，计算各顶点入度时需要遍历整个矩阵，因此时间复杂度为 $O(n^2)$，建立 0 入度顶点队列的时间复杂度为 O(n)。在拓扑排序过程中，(图中无环情况下)每个顶点进出队列各 1 次，实现入度减 1 操作时需遍历每个顶点的行向量 1 遍(时间复杂度为 O(n))，所以总的时间复杂度为 $O(n^2)$。

答案：

【问题 1】

(1) InitQueue(&Q)。

(2) DeQueue(&Q, &w)。

(3) inDegree[p->adjvex] 或其等价形式。

(4) inDegree[p->adjvex] 或其等价形式。

(5) k<G.n 或 k!=G.n 或其等价形式。

【问题 2】

队列方式：v1 v2 v5 v4 v3 v7 v6 或者 1 2 5 4 3 7 6。

栈方式：v1 v2 v5 v4 v7 v3 v6 或者 1 2 5 4 7 3 6。

【问题 3】

(6) O(n+e)。　　　　(7) $O(n^2)$。

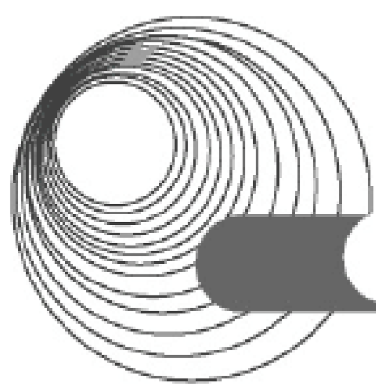
例 8　阅读下列说明和 C 代码，回答问题 1~问题 3，将解答写在答题纸的对应栏内。(2015 年 11 月试题四)

【说明】

计算两个字符串 x 和 y 的最长公共子串(Longest Common Substring)。

假设字符串 x 和字符串 y 的长度分别为 m 和 n，用数组 c 的元素 c[i][j]记录 x 中前 i 个字符和 y 中前 j 个字符的最长公共子串的长度。

c[i][j]满足最优子结构，其递归定义为：



$$c[i][j]=\begin{cases} c[i-1][j-1]+1 & 若 i>0 且 j>0 且 x[i]=y[j] \\ 0 & 其他 \end{cases}$$

计算所有 c[i][j](0≤i≤m，0≤j≤n)的值，值最大的 c[i][j]即为字符串 x 和 y 的最长公共子串的长度。根据该长度即 i 和 j，确定一个最长公共子串。

【C 代码】

(1) 常量和变量说明。

x，y：长度分别为 m 和 n 的字符串。

c[i][j]：记录 x 中前 i 个字符和 y 中前 j 个字符的最长公共子串的长度。

max：x 和 y 的最长公共子串的长度。

maxi, maxj：分别表示 x 和 y 的某个最长公共子串的最后一个字符在 x 和 y 中的位置(序号)。

(2) C 程序。

```
#include <stdio.h>
#include <string.h>
int c[50][50];
int maxi;
int maxj;
int lcs(char *x, int m, char *y, int n)     {
    int i, j;
    int max=0;
   maxi=0;
   maxj=0;
for (i=0; i<=m; i++)            c[i][0]=0;
for (i =1; i<=n; i++)           c[i][0]=0;
for (i =1; i<=m; i++)    {
     for (j=1; j<=n; j++)    {
        if (____(1)____)     {
c[i][j]=c[i-1][j-1]+1;
if(max<c[i][j]) {
 ______(2)____;
    maxi=i;
    maxj=j;
  }
}
else ____(3)____;
        }
     }
    return max;
}
void printLCS(int max, char *x) {
         int i=0;
      if (max==0)       return;
     for (____(4)____ ; i<maxi; i++)
printf("%c",x[i]);
}
```

```
void main(){
  char*x="ABCADAB";
  char*y="BDCABA";
  int max=0;
  int m=strlen(x);
  int n=strlen(y);
  max=lcs(x,m,y,n);
  printLCS(max,x);
}
```

【问题 1】

根据以上说明和 C 代码，填充 C 代码中的空(1)~(4)。

【问题 2】

根据题干说明和以上 C 代码，算法采用了__(5)__设计策略。

分析时间复杂度为__(6)__(用 O 符号表示)。

【问题 3】

根据题干说明和以上 C 代码，输入字符串 x=“ABCADAB”，y=“BDCABA”，则输出为__(7)__。

解析：

【问题 1】

这是一道非常经典的动态规划题，需要对动态规划的概念以及算法有一定的了解。考虑最长公共子串问题如何分解成子问题。假设 X=“$x_0,x_1,\cdots,x_{m-1}$”，Y=“$y_0,y_1,\cdots y_{n-1}$”，C=“$c_0,c_1,\cdots,c_{k-1}$”为它们的最长公共子序列。不难证明有以下性质：

(1) 如果 $x_{m-1}==y_{n-1}$，则 $c_{k-1}=x_{m-1}=y_{n-1}$，且“c_0，c_1，…，c_{k-2}”是“x_0，x_1，…，x_{m-2}”和“y_0，y_1，…，y_{n-2}”的一个最长公共子串。

(2) 如果 $x_{m-1}!=y_{n-1}$，则若 $c_{k-1}!=x_{m-1}$ 时，蕴涵“c_0，c_1，…，c_{k-1}”是“x_0，x_1，…，x_{m-2}”和“y_0，y_1，…，y_{n-1}”的一个最长公共子串。

(3) 如果 $x_{m-1}!=y_{n-1}$，则若 $c_{k-1}!=y_{n-1}$ 时，蕴涵“c_0，c_1，...，c_{k-1}”是“x_0，x_1，...，x_{m-1}”和“y_0，y_1，…，y_{n-2}”的一个最长公共子串。

这样，在找 x 和 y 的公共子串时，如果有 $x_{m-1}==y_{n-1}$，则进一步解决一个子问题，找“x_0，x_1，…，x_{m-2}”和“y_0，y_1，…，y_{m-2}”的一个最长公共子串；如果 $x_{m-1}!=y_{n-1}$，则要解决两个子问题，找出“x_0，x_1，…，x_{m-2}”和“y_0，y_1，…，y_{n-1}”的一个最长公共子串和找出“x_0，x_1，…，x_{m-1}”和“y_0，y_1，…，y_{n-2}”的一个最长公共子串，再取两者中较长者作为 x 和 y 的最长公共子串。

将空(1)~(4)补充完整的代码如下：

```
for (i=0; i<=m; i++)          c[i][0]=0;
for (i=1; i<=n; i++)          c[0][i]=0;
for (i=1; i<=m; i++)    {
   for (j=1; j<=n; j++)    {
      if (x[i-1]==y[j-1])    {
         c[i][j]=c[i-1][j-1]+1;
         if(max<c[i][j]) {
            max=c[i][j];
```

```
                maxi=i;
                maxj=j;
             }
           }
           else
              c[i][j]=0;
        }
        return max;
     }
     void printLCS(int max, char *x) {
        int i=0;
        if (max==0)   return;
        for (i=maxi-max; i<maxi; i++)
           printf("%c",x[i]);
     }
```

【问题 2】

很明显，该题目的解法采用了动态规划法的设计策略；由于嵌套了两次循环，所以该算法的时间复杂度为 O(m*n)。

【问题 3】

根据上述算法，列出两个字符串的 x 和 y 产生的 c[i][j]数据如下：

0 0 0 0 0 0 0
0 0 0 0 1 0 1
0 1 0 0 0 2 0
0 0 0 1 0 0 0
0 0 0 0 2 0 1
0 0 1 0 0 0 0
0 0 0 0 1 0 1
0 1 0 0 0 2 0

从数据中可以看出，最大子串长度是 2，位置为[2,5]、[4,4]、[7,5]，对应的字符串分别是 AB,CA,AB。根据代码描述，如果后面的子串长度不大于之前的子串，不进行替换，最终输出结果为 AB。

答案：

【问题 1】

(1) x[i−1] == y[j−1]。

(2) max=c[i][j]。

(3) c[i][j]=0。

(4) i=maxi−max。

【问题 2】

(5) 动态规划法。

(6) O(m*n)。

【问题 3】

(7) AB。

5.1.3　同步练习

1. 阅读下列说明，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。(2009 年 11 月试题四)

【说明】

0-1 背包问题可以描述为：有 n 个物品，对 i=1, 2, …, n，第 i 个物品价值为 v_i，重量为 w_i(v_i 和 w_i 为非负数)，背包容量为 W(W 为非负数)，选择其中一些物品装入背包，使装入背包物品的总价值最大，即 $\max\sum_{i=1}^{n}v_ix_i$，且总重量不超过背包容量，即 $\sum_{i=1}^{n}w_ix_i\leqslant W$，其中，$x_i\in\{0,1\}$，$x_i$=0 表示第 i 个物品不放入背包，$x_i$=1 表示第 i 个物品放入背包。

【问题 1】

用回溯法求解此 0-1 背包问题，请填充下面伪代码中(1)~(4)处空缺。

回溯法是一种系统的搜索方法。在确定解空间后，回溯法从根节点开始，按照深度优先策略遍历解空间树，搜索满足约束条件的解。对每一个当前节点，若扩展该节点已经不满足约束条件，则不再继续扩展。为了进一步提高算法的搜索效率，往往需要设计一个限界函数，判断并剪枝那些即使扩展了也不能得到最优解的节点。现在假设已经设计了 BOUND(v,w,k,W)函数，其中 v、w、k 和 W 分别表示当前已经获得的价值、当前背包的重量、已经确定是否选择的物品数和背包的总容量。对应于搜索树中的某个节点，该函数值表示确定了部分物品是否选择之后，对剩下的物品在满足约束条件的前提下进行选择可能获得的最大价值，若该价值小于等于当前已经得到的最优解，则该节点无须再扩展。

下面给出 0-1 背包问题的回溯算法的伪代码。

函数参数说明如下。

W：背包容量；n：物品个数；w：重量数组；v：价值数组；fw：获得最大价值时背包的重量；fp：背包获得的最大价值；X：问题的最优解。

变量说明如下。

cw：当前的背包重量；cp：当前获得的价值；k：当前考虑的物品编号；Y：当前已获得的部分解。

```
BKNAP(W,n,w,v,fw,fp,X)
1  cw ← cp ← 0
2  ___(1)___
3  fp ← -1
4  while true
5      while k≤n and cw+w[k] ≤W do
6                  ___(2)___
7                  cp ← cp+v[k]
8                  Y[k] ← 1
9                  k ← k+1
10     if k>n then
11         if fp<cp then
12             fp ← cp
13             fw ← cw
```

```
14              k ← n
15              X ← Y
16      else Y(k) ← 0
17      while BOUND(cp,cw,k,Y)≤fp do
18          while k≠0 and Y(k)≠1 do
19              (3)
20          if k=0 then return
21          Y[k]←0
22          cw ← cw-w[k]
23          cp ← cp-v[k]
24      (4)
```

【问题 2】

考虑表 5-4 所示的实例，假设有 3 个物品，背包容量为 22。图 5-6 是根据上述算法构造的搜索树，其中节点的编号表示了搜索树生成的顺序，边上的数字 1/0 分别表示选择/不选择对应物品。除了根节点之外，每个左孩子节点旁边的上、下两个数字分别表示当前背包的重量和已获得的价值，右孩子节点旁边的数字表示扩展了该节点后最多可能获得的价值。为获得最优解，应该选择物品__(5)__，获得的价值为__(6)__。

表 5-4　0-1 背包问题实例

属　性	物品信号		
	物品 1	物品 2	物品 3
重量	15	10	10
价值	30	18	17
单位价值	2	1.8	1.7

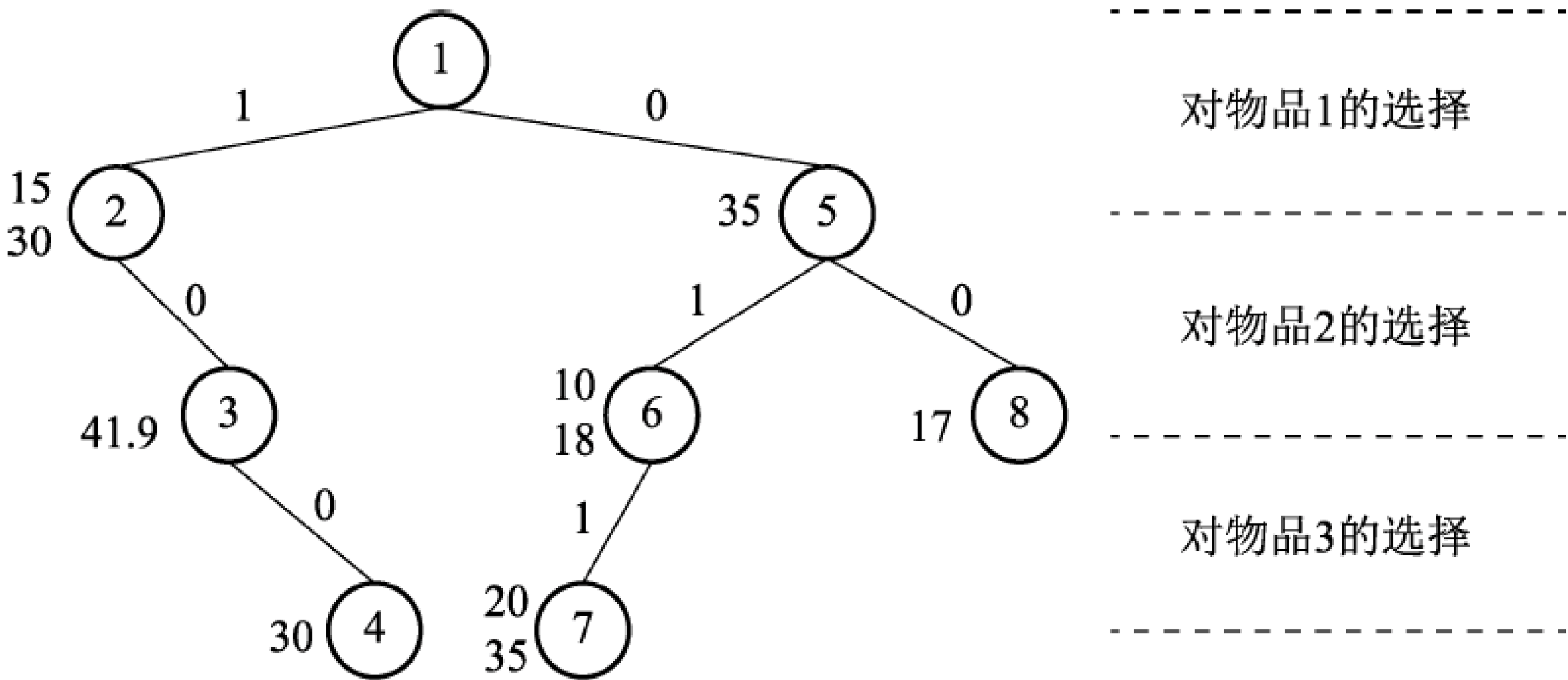


图 5-6　表 5-4 所示实例的搜索树

对于表 5-4 所示的实例，若采用穷举法搜索整个解空间，则搜索树的节点数为__(7)__，而用了上述回溯法后，搜索树的节点数为__(8)__。

2. 阅读以下说明和 C 程序，将应填入 (n) 处的子句写在答题纸的对应栏内。(2009 年 11 月试题七)

【说明】

现有 n(n<1000)节火车车厢，顺序编号为 1,2,3,…,n，按编号连续依次从 A 方向的铁轨

驶入，从 B 方向铁轨驶出，一旦车厢进入车站(Station)就不能再回到 A 方向的铁轨上；一旦车厢驶入 B 方向铁轨就不能再回到车站，如图 5-7 所示，其中 Station 为栈结构，初始为空且最多能停放 1000 节车厢。

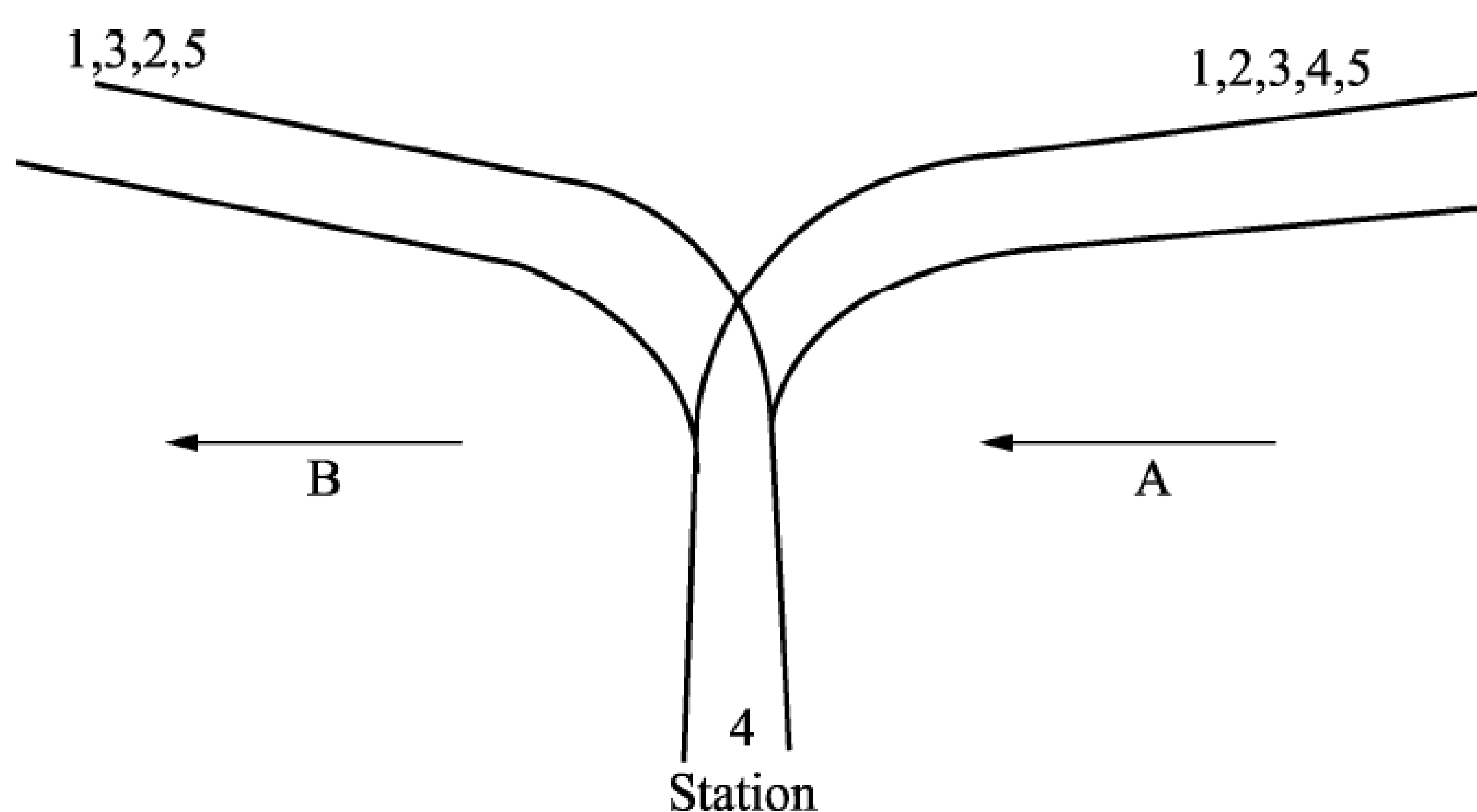


图 5-7　车站示意图

下面的 C 程序判断能否从 B 方向驶出预先指定的车厢序列，程序中使用了栈类型 STACK，关于栈基本操作的函数原型说明如下。

- void InitStack(STACK *s)：初始化栈。
- void Push (STACK *s, int e)：将一个整数压入栈，栈中元素数目增 1。
- void Pop (STACK *s)：栈顶元素出栈，栈中元素数目减 1。
- int Top (STACK s)：返回非空栈的栈顶元素值，栈中元素数目不变。
- int IsEmpty (STACK s)：若是空栈则返回 1；否则返回 0。

【C 程序】

```
#include <stdio.h>
/*此处为栈类型及其基本操作的定义，省略*/
 int main(){
        STACK station;
        int state[1000];
        int n;                          /*车厢数*/
        int begin, i, j, maxNo;  /*maxNo 为 A 端正待入栈的车厢编号*/
        printf("请输入车厢数：");
        scanf("%d",&n);
        printf("请输入需要判断的车厢编号序列(以空格分隔)：");
        if(n<1)
             return -1;
        for (i=0; i<n; i++)        /*读入需要驶出的车厢编号序列，存入数组 state[]*/
             scanf("%d",&state[i]);
        __(1)__;                        /*初始化栈*/
        maxNo=1;
        for(i=0; i<n; {           /*检查输出序列中的每个车厢号 state[i]是否能从栈中获取*/
             if(__(2)__){          /*当栈不为空时*/
                 if (state[i]=Top(station)) {      /*栈顶车厢号等于被检查车厢号*/
                           printf("%d",Top(station));
                           Pop(&station);i++;
                 }
```

```
                    else
                    if (  (3)  ) {
                        printf("error\n");
                         return 1;
                    }
                    else {
                         begin=  (4)  ;
                         for(j=begin+1;j<=state[i];j++) {
                                Push(&station, j);
                           }
                     }
              }
              else {    /*当栈为空时*/
                    begin=maxNo;
                   for(j=begin; j<=state[i]; j++) {
                         Push(&station, j);
             }
             maxNo=  (5)  ;
             }
        }
        printf("OK");
        return 0;
    }
```

3. 阅读下列说明，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。(2009 年 5 月试题四)

【说明】

现需在某城市中选择一个社区建一个大型超市，使该城市的其他社区到该超市的距离总和最小。用图模型表示该城市的地图，其中顶点表示社区，边表示社区间的路线，边上的权重表示该路线的长度。

现设计一个算法来找到该大型超市的最佳位置，即在给定图中选择一个顶点，使该顶点到其他各顶点的最短路径之和最小。算法首先需要求出每个顶点到其他任一顶点的最短路径，即需要计算任意两个顶点之间的最短路径；然后对每个顶点，计算其他各顶点到该顶点的最短路径之和；最后，选择最短路径之和最小的顶点作为建大型超市的最佳位置。

【问题 1】

本题采用 Floyd-Warshall 算法求解任意两个顶点之间的最短路径。已知图 G 的顶点集合为 V={1,2,⋯,n}，**W**= $\{w_{ij}\}_{n\times n}$ 为权重矩阵。设 $d_{ij}^{(k)}$ 为从顶点 i 到顶点 j 的一条最短路径的权重。当 k = 0 时，不存在中间顶点，因此 $d_{ij}^{(0)}=w_{ij}$。

当 k>0 时，该最短路径上所有的中间顶点均属于集合{1,2,⋯, k}。若中间顶点包括顶点 k，则 $d_{ij}^{(k)}=d_{ik}^{(k-1)}+d_{kj}^{(k-1)}$；若中间顶点不包括顶点 k，则 $d_{ij}^{(k-1)}=d_{i}^{(k-1)}$。

于是得到如下递归式：

$$d_{ij}^{(k)}=\begin{cases} w_{ij} & k=0 \\ \min(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)}+d_{kj}^{(k-1)}) & k>0 \end{cases}$$

因为对于任意路径，所有的中间顶点都在集合{1,2, ⋯, n}内，因此矩阵 $\mathbf{D}^{(n)}=\{d_{ij}^{(n)}\}_{n\times n}$

给出了任意两个顶点之间的最短路径，即对所有 i, j ∈V，$d_{ij}^{(n)}$ 表示顶点 i 到顶点 j 的最短路径。

下面是求解该问题的伪代码，请填充其中空缺的(1)~(6)处。伪代码中的主要变量说明如下。

- W：权重矩阵。
- n：图的顶点个数。
- SP：最短路径权重之和数组，SP[i]表示顶点 i 到其他各顶点的最短路径权重之和，i 从 1 到 n。
- min_SP：最小的最短路径权重之和。
- min_v：具有最小的最短路径权重之和的顶点。
- i：循环控制变量。
- j：循环控制变量。
- k：循环控制变量。

```
LOCATE-SHOPPINGMALL(W, n)
1    D(0)=W
2    for      (1)
3             for i = 1 to n
4                      for j = 1 to n
5                      if d_ij^(k-1) ≤ d_ik^(k-1) + d_kj^(k-1)
6                                                  (2)
7                                  else
8                                                  (3)
9    for i = 1 to n
10             SP[i] = 0
11             for j = 1 to n
12                          (4)
13    min_SP = SP[1]
14       (5)
15    for i = 2 to n
16            if min_SP > SP[i]
17                   min_SP = SP[i]
18                   min_v = i
19   return      (6)
```

【问题 2】

问题 1 中伪代码的时间复杂度为__(7)__(用 O 符号表示)。

4. 阅读下列说明和 C 函数代码，将应填入(n)处的子句写在答题纸的对应栏内。(2009 年 5 月试题五)

【说明】

对二叉树进行遍历是二叉树的一个基本运算。遍历是指按某种策略访问二叉树的每个节点，且每个节点仅访问一次的过程。函数 InOrder()借助栈实现二叉树的非递归中序遍历运算。

设二叉树采用二叉链表存储，节点类型定义如下：

```
typedef struct BtNode{
        ElemType    data;         /*节点的数据域，ElemType 的具体定义省略*/
        struct BtNode *lchild,*rchild;       /*节点的左、右孩子指针域*/
}BtNode, *BTree;
```

在函数 InOrder()中，用栈暂存二叉树中各个节点的指针，并将栈表示为不含头节点的单向链表(简称链栈)，其节点类型定义如下：

```
typedef struct StNode{          /*链栈的节点类型*/
        BTree elem;             /*栈中的元素是指向二叉链表节点的指针*/
        struct StNode *link;
}StNode;
```

假设从栈顶到栈底的元素为 e_n, e_{n-1}, …, e_1，则不含头节点的链栈示意图如图 5-8 所示。

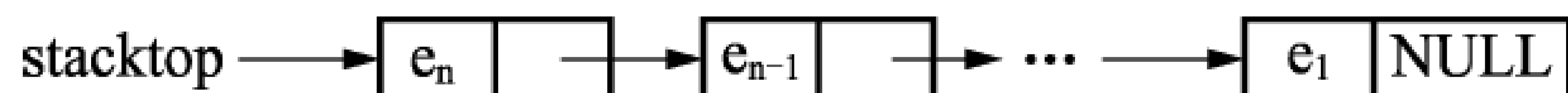


图 5-8　链栈示意图

【C 程序】

```
int InOrder(BTree root)                /*实现二叉树的非递归中序遍历*/
{
    BTree ptr;                         /*ptr 用于指向二叉树中的节点*/
    StNode *q;                         /*q 暂存链栈中新创建或待删除的节点指针*/
    StNode *stacktop = NULL;           /*初始化空栈的栈顶指针 stacktop*/
    ptr = root;                        /*ptr 指向二叉树的根节点*/
    while (____(1)____|| stacktop != NULL)
    {
       while (ptr != NULL)
    {
           q = (StNode *)malloc(sizeof(StNode));
           if (q == NULL)
               return -1;
           q→elem = ptr;
         ___(2)___;
          stacktop = q;                /*stacktop 指向新的栈顶*/
          ptr = ___(3)___;             /*进入左子树*/
    }
       q = stacktop;
       ___(4)___;                      /*栈顶元素出栈*/
       visit(q);                       /*visit 是访问节点的函数，其具体定义省略*/
       ptr =___(5)___;                 /*进入右子树*/
       free(q);                        /*释放原栈顶元素的节点空间*/
    }
    return 0;
  }/*InOrder*/
```

5. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2008 年 12 月试题四)

【说明】

某餐厅供应各种标准的营养套餐。假设菜单上共有n项食物$m_1,m_2,\cdots,m_n$，每项食物m_i的营养价值为v_i，价格为p_i，其中$i=1,2,\cdots,n$，套餐中每项食物至多出现一次。客人常需要一个算法来求解总价格不超过M的营养价值最大的套餐。

【问题1】

下面是用动态规划策略求解该问题的伪代码，请填充其中的空缺(1)~(3)处。伪代码中的主要变量说明如下。

n：总食物项数。

v：营养价值组，下标为 1~n，对应第 1~n 项食物的营养价值。

p：价格数组，下标为 1~n，对应第 1~n 项食物的价格。

M：总标准，即套餐的价格不超过 M。

x：解向量(数组)，下标为 1~n，其元素值为 0 或 1，其中元素值为 0 表示对应的食物不出现在套餐中，元素值为 1 表示对应的食物出现在套餐中。

nv：n+1 行 M+1 列的二维数组，其中行和列的下标均从 0 开始，nv[i][j]表示由前 i 项食物组合且价格不超过 j 套餐的最大营养价值。问题最终要求套餐的最大营养价值为nv[n][M]。伪代码如下：

```
MaxNutrientValue(n, v, p, M, x)
1   for i = 0 to n
2      nv[i][0] = 0
3   for j = 1 to M
4      nv[0][j] = 0
5   for i = 1 to n
6    for j = 1 to M
7        if j < p[i] //若食物mi不能加入到套餐中
8           nv[i][j] = nv[i - 1][j]
9       else if  (1)  
10          nv[i][j] = nv[i - 1][j]
11      else
12          nv[i][j] = nv[i - 1][j - p[i]] + v[i]
13   j = M
14   for i = n downto 1
15     if  (2)  
16        x[i] = 0
17     else
18        x[i] = 1
19        (3)  
20   return x and nv[n][M]
```

【问题 2】

现有 5 项食物，每项食物的营养价值和价格如表 5-5 所示。

表 5-5　食物营养价值及价格

编　码	营养价值	价　格
m_1	200	50
m_2	180	30
m_3	225	45
m_4	200	25
m_5	50	5

若要求总价格不超过 100 元的营养价值最大的套餐，则套餐应包含的食物有__(4)__(用食物项的编码表示)，对应的最大营养价值为__(5)__。

【问题 3】问题 1 中伪代码的时间复杂度为__(6)__(用 O 符号表示)。

6. 阅读下列说明和 C 函数代码，将应填入(n)处的子句写在答题纸的对应栏内。(2008 年 12 月试题五)

【说明】

已知集合 A 和 B 的元素分别用不含头节点的单链表存储，函数 Difference()用于求解集合 A 与 B 的差集，并将结果保存在集合 A 的单链表中。例如，若集合 A={5,10,20,15,25,30}，集合 B={5,15,35,25}，如图 5-9(a)所示，运算完成后的结果如图 5-9(b)所示。

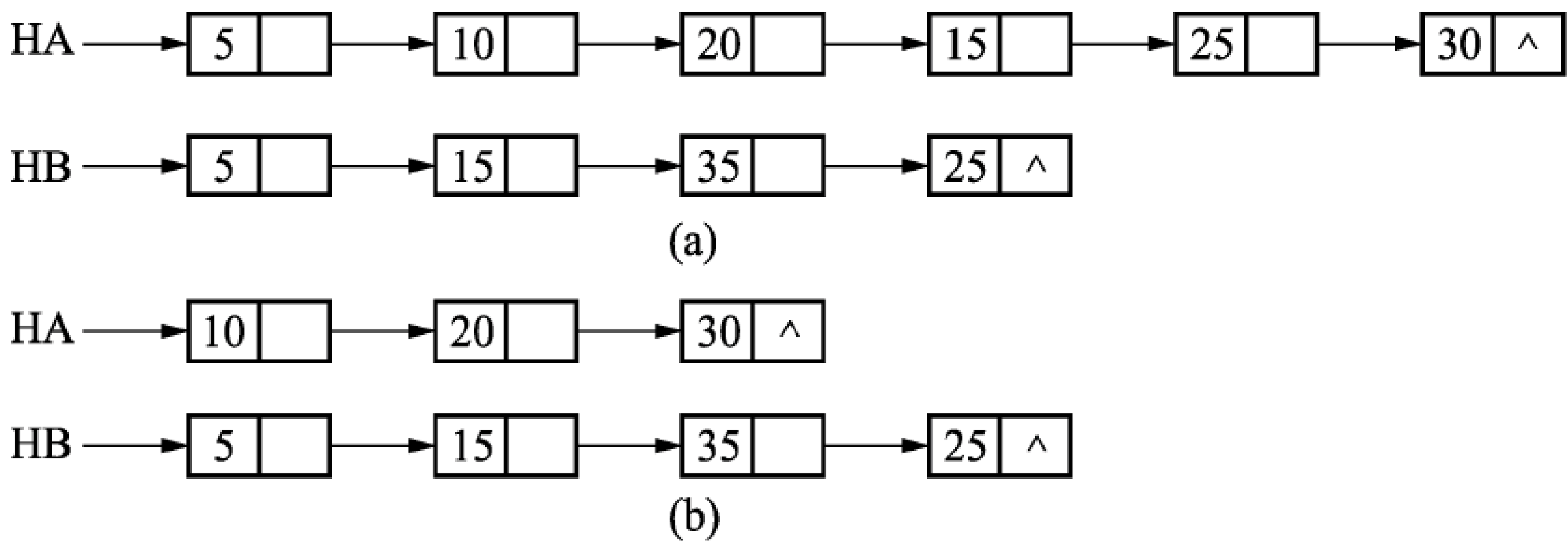


图 5-9　集合 A、B 运算前后示意图

链表节点的结构类型定义如下：

```
typedef struct Node{
    ElemType elem;
    struct Node *next;
}NodeType;
```

【C 程序】

```
void Difference(NodeType **LA, NodeType *LB)
{
    NodeType *pa, *pb, *pre, *q;
    pre = NULL;
    (1);
    while (pa)
   {
```

```
        pb = LB;
        while (_(2)_)
        pb = pb→next;
        if (_(3)_)
        {
           if (!pre)
              *LA = _(4)_;
           else
              _(5)_= pa→next;
           q = pa;
           pa = pa→next;
           free(q);
        }
       else {
           _(6)_;
           pa = pa→next;
       }
    }
}
```

7. 阅读下列说明，回答问题 1~问题 3，将解答填入答题纸的对应栏内。(2008 年 5 月试题四)

【说明】

快速排序是一种典型的分治算法。采用快速排序对数组 A[p..r]排序的 3 个步骤如下。

(1) 分解：选择一个枢轴(Pivot)元素划分数组。将数组 A[p..r]划分为两个子数组(可能为空)A[p..q−1]和 A[q+1..r]，使得 A[q]大于等于 A[p..q−1]中的每个元素，小于 A[q+1..r]中的每个元素。q 的值在划分过程中计算。

(2) 递归求解：通过递归调用快速排序，对子数组 A[p..q−1]和 A[q+1..r]分别排序。

(3) 合并：快速排序在原地排序，故不需合并操作。

【问题 1】

下面是快速排序的伪代码，请填补其中的空缺。伪代码中的主要变量说明如下。

- A：待排序数组。
- p, r：数组元素下标，从 p~r。
- q：划分的位置。
- x：枢轴元素。
- i：整型变量，用于描述数组下标。下标小于或等于 i 的元素的值小于或等于枢轴元素的值。
- j：循环控制变量，表示数组元素下标。

```
QUICKSORT(A, p, r){
    if (p < r){
        q = PARTITION(A,p,r) ;
        QUICKSORT(A, p, q-1);
        QUICKSORT(A, q+1, r);
```

```
        }
    }
    PARTITION(A, p, r){
        x = A[r]; i = p - 1;
        for (j = p; j≤r - 1; j++){
            if (A[j]≤x){
                i = i + 1 ;
                交换 A[i]和 A[j]
            }
        }
        交换 (1) 和 (2) //注：空(1)和空(2)答案可互换，但两空全部答对方可得分
        return (3)
    }
```

【问题 2】

(1) 假设要排序包含 n 个元素的数组，请给出在各种不同的划分情况下，快速排序的时间复杂度，用 O 标记。最佳情况为 __(4)__ ，平均情况为 __(5)__ ，最坏情况为 __(6)__ 。

(2) 假设要排序的 n 个元素都具有相同值时，快速排序的运行时间复杂度属于哪种情况？ __(7)__ 。(最佳、平均、最坏)

【问题3】

(1) 待排序数组是否能被较均匀地划分对快速排序的性能有重要影响，因此枢轴元素的选取非常重要。有人提出从待排序的数组元素中随机地取出一个元素作为枢轴元素，下面是随机化快速排序划分的伪代码——利用原有的快速排序的划分操作，请填充其中的空缺处。其中，RANDOM(i, j)表示随机取 i~j 的一个数，包括 i 和 j。

```
    RANDOMIZED-PARTITION(A,p,r){
        i = RANDOM(p,r);
        交换 (8) 和 (9) ;//注：空(8)和空(9)答案可互换，但两空全部答对方可得分
        return PARTITION(A,p,r);
    }
```

(2) 随机化快速排序是否能够消除最坏情况的发生？ __(10)__ 。(是或否)

8. 阅读下列说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2008 年 5 月试题五)

【说明】

栈(Stack)结构是计算机语言实现中的一种重要数据结构。对于任意栈，进行插入和删除操作的一端称为栈顶(Stack Top)，而另一端称为栈底(Stack Bottom)。栈的基本操作包括创建栈(NewStack)、判断栈是否为空(IsEmpty)、判断栈是否已满(IsFull)、获取栈顶数据(Top)、压栈/入栈(Push)、弹栈/出栈(Pop)。

当设计栈的存储结构时，可以采取多种方式。其中，采用链式存储结构实现的栈中各数据项不必连续存储，如图 5-10 所示。

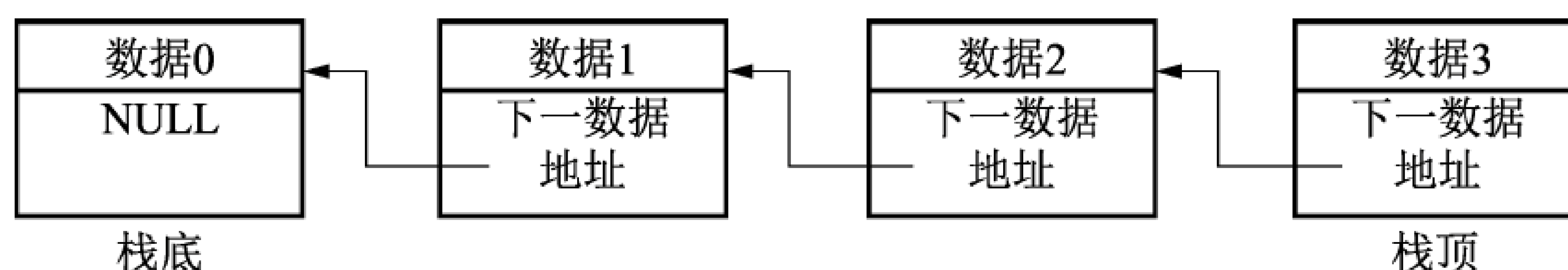


图 5-10　栈的链式存储结构示意图

以下 C 代码采用链式存储结构实现一个整数栈操作。

【C 程序】

```
typedef struct List
{
    int data;                //栈数据
    struct List* next;       //上次入栈的数据地址
}List;

typedef struct Stack{
    List* pTop;              //当前栈顶指针
}Stack;

Stack* NewStack(){return (Stack*)calloc(1,sizeof(Stack));}

int IsEmpty(Stack* S)        //判断栈 S 是否为空栈
{
    If( (1) ) return 1;
    return 0;
}

int Top(Stack* S)            //获取栈顶数据。若栈为空，则返回机器可表示的最小整数
{
    if(IsEmpty(S)) return INT_MIN;
    return (2) ;
}

Void Push(Stack* S, int theData)         //将数据 theData 压栈
{
    List* newNode;
    newNode=(List*)calloc(1,sizeof(List));
    newNode→data=theData;
    newNode→next=S→pTop;
    S→pTop= (3) ;
}

Void Pop(Stack* S)           //弹栈
{
    List* lastTop;
    If(IsEmpty(S)) return;
    lastTop=S→pTop;
    S→pTop= (4) ;
```

```
    Free(lastTop);
)

#define MD(a) a<<2
int main()
{
    int i;
    Stack* myStack;
    myStack=NewStack();
    Push(myStack,MD(1));
    Push(myStack,MD(2));
    Pop(myStack);
    Push(myStack,MD(3)+1);
    while(!IsEmpty(myStack))
    {
            printf("%d",Top(myStack));
            Pop(myStack);
    }
    return ();
}
```

以上程序运行时的输出结果为__(5)__。

9. 阅读以下说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2007 年 11 月试题五)

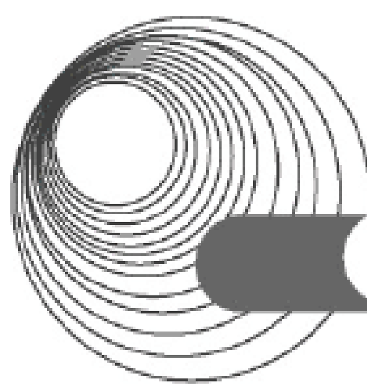
【说明】

在一个简化的绘图程序中，支持的图形种类有点(point)和圆(circle)，在设计过程中采用面向对象思想，认为所有的点和圆都是一种图形(shape)，并定义了类型 shape_t、point_t 和 circle_t 分别表示基本图形、点和圆，并且点和圆具有基本图形的所有特征。

【C 程序】

```
typedef enum { point,circle } shape_type;   /* 程序中的两种图形：点和圆 */
typedef struct {                /* 基本的图形类型 */
    shape_type  type;           /* 图形种类标识：点或者圆 */
    void  (*destroy)();         /* 销毁图形操作的函数指针 */
    void  (*draw)();            /* 绘制图形操作的函数指针 */
} shape_t;
typedef struct { shape_t common;  int x;  int y; } point_t;
/* 定义点类型，x、y 为点坐标 */
void destroyPoint(point_t* this) { free(this); printf("Point
destoryed!\n"); }  /* 销毁点对象 */
void drawPoint(point_t* this) { printf("P(%d,%d)", this→x, this→y); }
/* 绘制点对象 */
shape_t* createPoint(va_list* ap)      /* 创建点对象，并设置其属性 */
 {
    point_t*  p_point;
    if( (p_point = (point_t*)malloc(sizeof(point_t))) == NULL ) return NULL;
    p_point→common.type = point;
```

```
    p_point→common.destroy = destroyPoint;
    p_point→common.draw = drawPoint;
    p_point→x = va_arg(*ap, int);       /* 设置点的横坐标 */
    p_point→y = va_arg(*ap, int);       /* 设置点的纵坐标 */
    return (shape_t*)p_point;          /* 返回点对象指针 */
}
typedef struct          /* 定义圆类型 */
{
    shape_t  common;
    point_t  *center;  /* 圆心点 */
    int   radius;      /* 圆半径 */
} circle_t;
void destroyCircle(circle_t* this){
    free(__(1)__);      free(this);     printf("Circle destoryed!\n");
}
void drawCircle(circle_t* this)
{
    printf("C(");
    __(2)__.draw( this→center );             /* 绘制圆心 */
    printf(",%d)", this→radius);
}
shape_t* createCircle(va_list* ap)          /* 创建一个圆，并设置其属性 */
{
    circle_t*  p_circle;
    if( (p_circle = (circle_t*)malloc(sizeof(circle_t))) == NULL ) return NULL;
    p_circle→common.type = circle;  p_circle→common.destroy = destroyCircle;
    p_circle→common.draw = drawCircle;
    __(3)__ = createPoint(ap);                /* 设置圆心 */
    p_circle→radius = va_arg(*ap, int);     /* 设置圆半径 */
    return  p_circle;
}
shape_t* createShape(shape_type st, ...)     /* 创建某一种具体的图形 */
{
    va_list   ap;                             /* 可变参数列表 */
    shape_t*  p_shape = NULL;
        __(4)__(ap, st);
    if( st == point ) p_shape = createPoint( &ap);    /* 创建点对象 */
    if( st == circle ) p_shape = createCircle(&ap);    /* 创建圆对象 */
    va_end(ap);
    return p_shape;
}
int main( )
{
    int i;                    /* 循环控制变量，用于循环计数 */
    shape_t* shapes[2];  /* 图形指针数组，存储图形的地址 */
    shapes[0] = createShape(point, 2, 3);        /* 横坐标为 2，纵坐标为 3 */
    shapes[1] = createShape(circle, 20, 40, 10);
    /* 圆心坐标(20,40)，半径为 10 */
    for(i=0; i<2; i++) { shapes[i]→draw(shapes[i]);  printf("\n");  }
```




```
    /* 绘制数组中图形 */
    for(i = 1; i >= 0; i--)  shapes[i]→destroy(shapes[i]);  /* 销毁数组中图形 */
    return 0;
}
```

运行结果如下：

```
P(2,3)
```
__(5)__
```
Circle destoryed!
Point destoryed!
```

10. 阅读以下说明和C语言函数，将应填入(n)处的子句写在答题纸的对应栏内。(2007年5月试题五)

【说明】

在一个分布网络中，资源(石油、天然气、电力等)可从生产地送往其他地方。在传输过程中，资源会有损耗。例如，天然气的气压会减少，电压会降低。我们将需要输送的资源信息称为信号。在信号从信源地送往消耗地的过程中，仅能容忍一定范围的信号衰减，称为容忍值。分布网络可表示为一个树型结构，如图5-11所示。信号源是树根，树中的每个节点(除了根)表示一个可以放置放大器的子节点，其中某些节点同时也是信号消耗点，信号从一个节点流向其子节点。

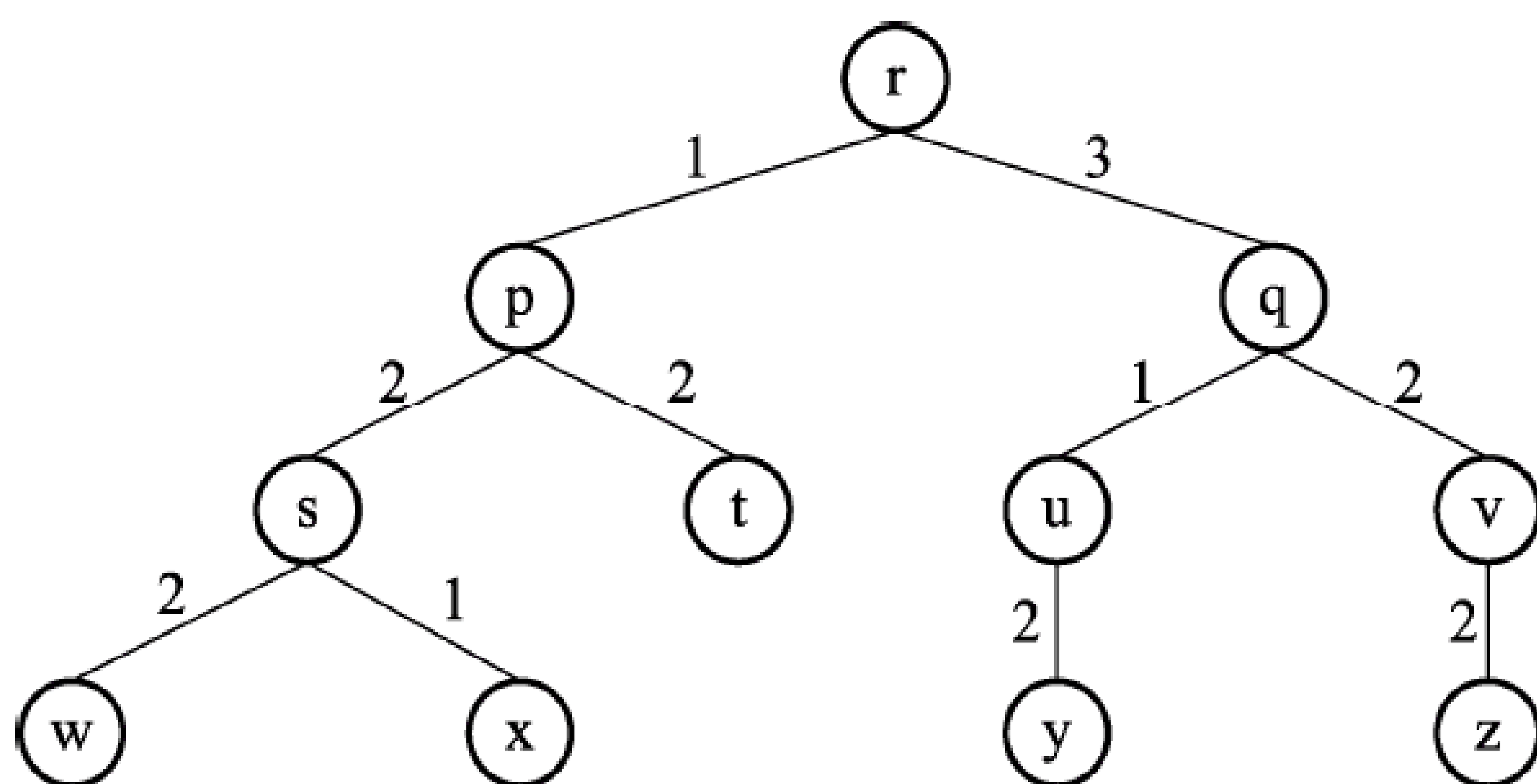


图 5-11　分布网络的树型结构表示

每个节点有一个d值，表示从其父节点到该节点的信号衰减量。例如，在图5-11中，节点w、p、q的d值分别为2、1、3，树根节点表示信号源，其d值为0。

每个节点有一个M值，表示从该节点出发到其所有叶子节点的信号衰减量的最大值。显然，叶子节点的M值为0。对于非叶子节点j，M(j)=max{M(k) + d(k) | k是j的子节点}。在此公式中，要计算节点的M值，必须先算出其所有子节点的M值。

在计算M值的过程中，对于某个节点i，若其有一个子节点k满足d(k)+M(k) 大于容忍值，则应在k处放置放大器，否则，从节点i到某叶子节点的信号衰减量会超过容忍值，使得到达该叶子节点时信号不可用，而在节点i处放置放大器并不能解决到达叶子节点的信号衰减问题。例如，在图5-11中，从节点p到其所有叶子节点的最大衰减值为4。若容忍值为3，则必须在s处放置信号放大器，这样可使得节点p的M值为2。同样，需要在节点q、v处放置信号放大器，如图5-12阴影节点所示。若在某节点放置了信号放大器，则从该节点输出的信号与信号源输出的信号等价。

函数 placeBoosters(TreeNode *root)的功能是：对于给定树型分布网络中的各个节点，计算其信号衰减量的最大值，并确定应在树中的哪些节点放置信号放大器。

全局变量 Tolerance 保存信号衰减容忍值。

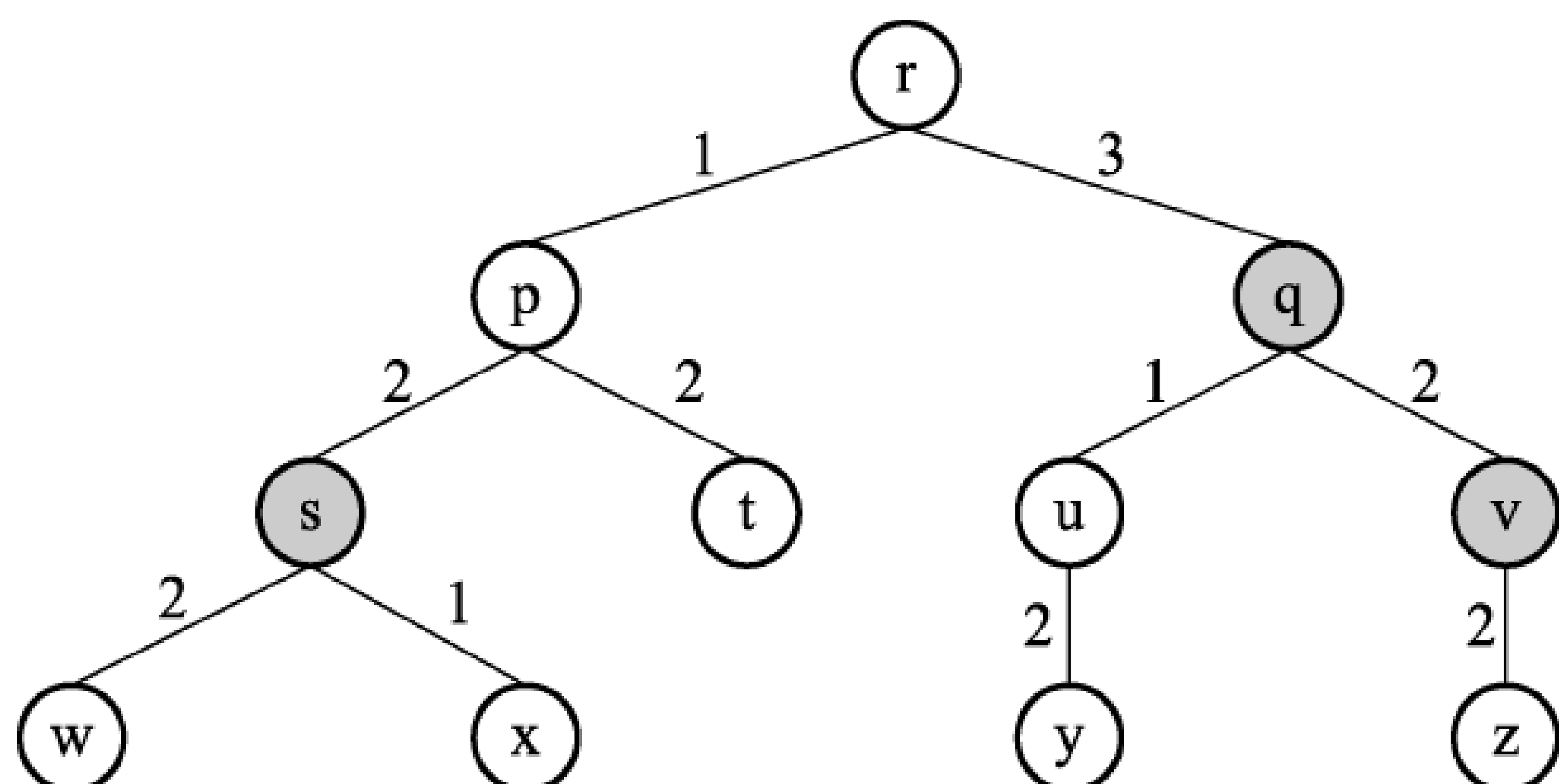


图 5-12　放置信号放大器的分布网络树型结构

树的节点类型定义如下：

```
typedef struct TreeNode {
int id; /*当前节点的识别号*/
int ChildNum; /*当前节点的子节点数目*/
int d; /*父节点到当前节点的信号衰减值*/
struct TreeNode **childptr; /*向量，存放当前节点到其所有子节点的指针*/
int M; /*当前节点到其所有子节点的信号衰减值中的最大值*/
bool boost; /*是否在当前节点放置信号放大器的标志*/
}TreeNode;
```

【C 程序】

```
void placeBoosters(TreeNode *root)
{   /*计算 root 所指节点处的衰减量，如果衰减量超出了容忍值，则放置放大器*/
    TreeNode *p;
    int i,degradation;
    if( (1) )
    {
           degradation=0;root→M=0;
           i=0;
           if(i>=root→ChildNum)
              return;
           p= (2) ;
           for(;i<root→ChildNum && p;i++,p= (3) )
           {
               p→M=0;
               (4) ;
               if(p→d+p→M>Tolerance)   /*在 p 所指节点中放置信号放大器*/
               {
                    p→boost=true;
                    p→M=0;
               }
               if(p→d+p→M>degradation)
```

```
                    degradation=p→d+p→M;
                }
                Root→M= (5) ;
            }
        }
```

11. 阅读以下说明、图和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2006 年 11 月试题五)

【说明】

一般的树型结构常采用孩子-兄弟表示法表示，即用二叉链表作树的存储结构，链表中节点的两个链域分别指向该节点的第一个孩子节点和下一个兄弟节点。例如，图 5-13(a)所示树的孩子-兄弟表示如图 5-13(b)所示。

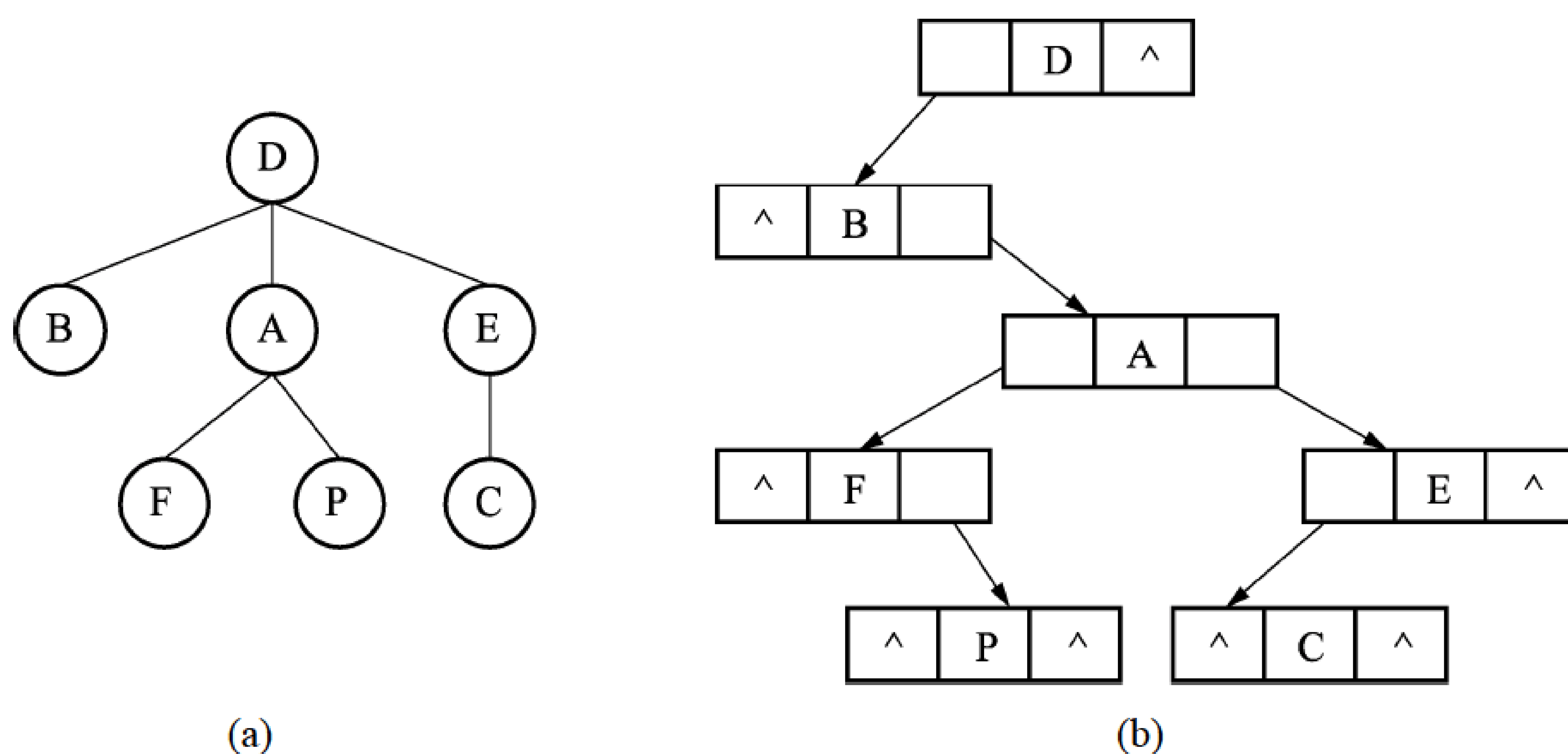


图 5-13　树及其孩子-兄弟表示示意图

函数 LevelTraverse()的功能是对给定树进行层序遍历。例如，对图 5-13 所示的树进行层序遍历时，节点的访问次序为 D B A E F P C。

对树进行层序遍历时使用了队列结构，实现队列基本操作的函数原型如表 5-6 所示。

表 5-6　实现队列基本操作的函数原型

函数原型	说　明
void InitQueue(Queue *Q)	初始化队列
bool IsEmpty(Queue Q)	判断队列是否为空，若是则返回 TRUE，否则返回 FALSE
void EnQueue(Queue *Q,TreeNode p)	元素入队列
void DeQueue(Queue *Q,TreeNode *p)	元素出队列

Bool、Status 类型定义如下：

```
typedef enum {FALSE = 0,TRUE = 1} Bool;
typedef enum {OVERFLOW = -2,UNDERFLOW = -1,ERROR = 0,OK = 1} Status;
```

树的二叉链表节点定义如下：

```
typedef struct Node {
  char data;
  struct Node *firstchild,*nextbrother;
}Node,*TreeNode;
```

函数 LevelTraverse 的代码如下：

```
Status LevelTraverse(TreeNode root)
{    /*层序遍历树，树采用孩子-兄弟表示法，root 是树根节点的指针*/
     Queue tempQ;
     TreeNode ptr,brotherptr;
     if (!root)
        return ERROR;
     InitQueue(&tempQ);
      (1) ;
     brotherptr = root→nextbrother;
     while (brotherptr)
     {
     EnQueue(&tempQ,brotherptr);
         (2) ;
     } /*end-while*/
    while ( (3) )
    {
        (4) ;
        printf("%c\t",ptr→data);
        if ( (5) ) continue;
         (6) ;
        brotherptr = ptr→firstchild→nextbrother;
        while (brotherptr)
        {
        EnQueue(&tempQ,brotherptr);
             (7) ;
        } /*end-while*/
    } /*end-while*/
    return OK;
}/*LevelTraverse*/
```

12. 阅读下列说明、图和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2006 年 5 月试题五)

【说明 1】

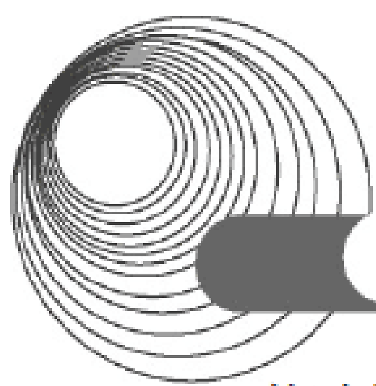
B 树是一种多叉平衡查找树。一棵 m 阶的 B 树，或为空树，或为满足下列特性的 m 叉树。

(1) 树中每个节点至多有 m 棵子树。

(2) 若根节点不是叶子节点，则它至少有两棵子树。

(3) 除根之外的所有非叶子节点至少有[m/2]棵子树。

(4) 所有的非叶子节点中包含下列数据信息：$(n,A_0,K_1,A_1,K_2,A_2,\cdots,K_n,A_n)$。其中，$K_i(i=1,2,\cdots,n)$为关键字，且 $K_i<K_i+1(i=1,2,\cdots,n-1)$，$A_i(i=0,1,\cdots,n)$为指向树根节点的指针，且



指针 A_{i-1} 所指子树中所有节点的关键字均小于 k_i，A_{i+1} 所指子树中所有节点的关键字均大于 k_i；n 为节点中关键字的数目。

(5) 所有的叶子节点都出现在同一层次上，并且不带信息(可以看作是外部节点或查找失败的节点，实际上这些节点不存在，指向这些节点的指针为空)。

例如，一棵 4 阶 B 树如图 5-14 所示(节点中关键字的数目省略)。

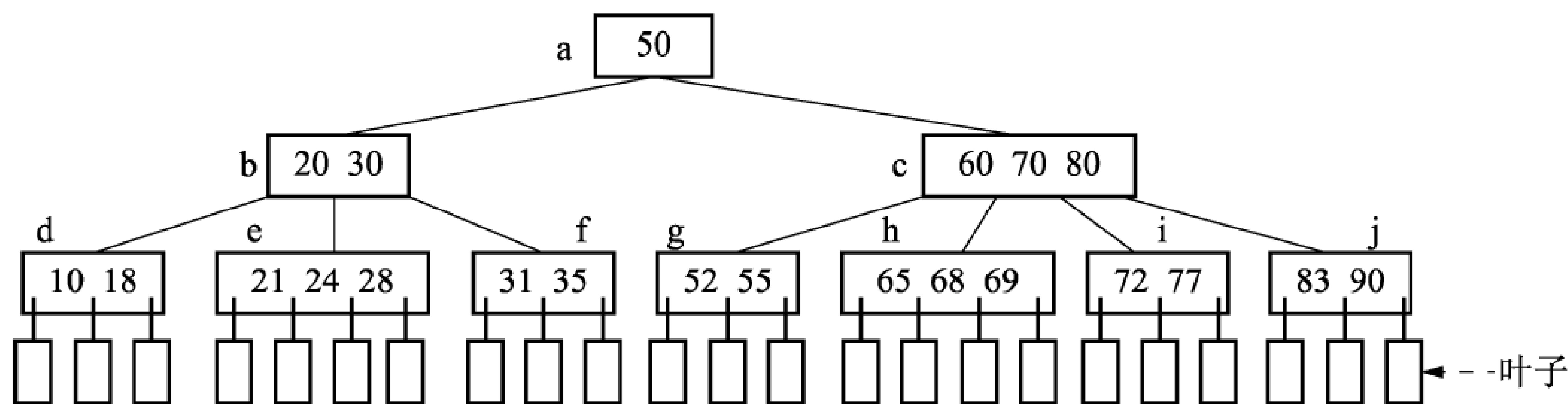


图 5-14　4 阶 B 树示例

B 树的阶 M、bool 类型、关键字类型及 B 树节点的定义如下：

```
#define  M  4                       /*B 树的阶*/
typedef enum {FALSE=0, TRUE = 1} bool;
typedef int ElemKeyType;
typedef struct BTreeNode{
  int numkeys;                      /*节点中关键字的数目*/
  struct BTreeNode *parent;         /*指向父节点的指针，树根的父节点指针为空*/
  struct BTreeNode *A[M];           /*指向子树节点的指针数组*/
  ElemgeyType K[M];                 /*存储关键字的数组，K[0]闲置不用*/
}BTreeNode;
```

函数 SearchBtree(BTreeNode* root,ElemKeyType akey,BTreeNode **ptr)的功能是：在给定的一棵 M 阶 B 树中查找关键字 akey 所在节点，若找到则返回 TRUE，否则返回 FALSE。其中，root 是指向该 M 阶 B 树根节点的指针，参数 ptr 返回 akey 所在节点的指针，若 akey 不在该 B 树中，则 ptr 返回查找失败时空指针所在节点的指针。例如，在图 5-14 所示的 4 阶 B 树中查找关键字 25 时，ptr 返回指向节点 e 的指针。

注：在节点中查找关键字 akey 时采用二分法。

函数 SearchBtree 的代码如下：

```
bool SearchBtree(BTreeNode* root, ElemKeyType akey, BTreeNode **ptr)
{
  int  lw, hi, mid;
  BTreeNode *p = root;
  *ptr = NULL;
  while(p){
      lw = 1; hi = __(1)__;
      while (lw <= hi) {
          mid = (lw + hi) / 2;
          if(p → K[mid] == akey){
              *ptr = p;
```

```
                return TRUE;
            }else if ( (2) )
                hi = mid - 1;
            else
                lw = mid + 1;
        }
        *ptr = p;
        p =  (3) ;
    }
    return FALSE;
}
```

【说明 2】

在 M 阶 B 树中插入一个关键字时，首先在最接近外部节点的某个非叶子节点中增加一个关键字，若该节点中关键字的个数不超过 M-1，则完成插入；否则，要进行节点的“分裂”处理。所谓“分裂”，就是把节点中处于中间位置上的关键字取出来并插入其父节点中，然后以该关键字为分界线，把原节点分成两个节点。“分裂”过程可能会一直持续到树根，若树根节点也需要分裂，则整棵树的高度增 1。

例如，在图 5-14 所示的 B 树中插入关键字 25 时，需将其插入节点 e 中。由于 e 中已经有 3 个关键字，因此将关键字 24 插入节点 e 的父节点 b 中，并以 24 为分界线将节点 e 分裂为 e_1 和 e_2 两个节点，结果如图 5-15 所示。

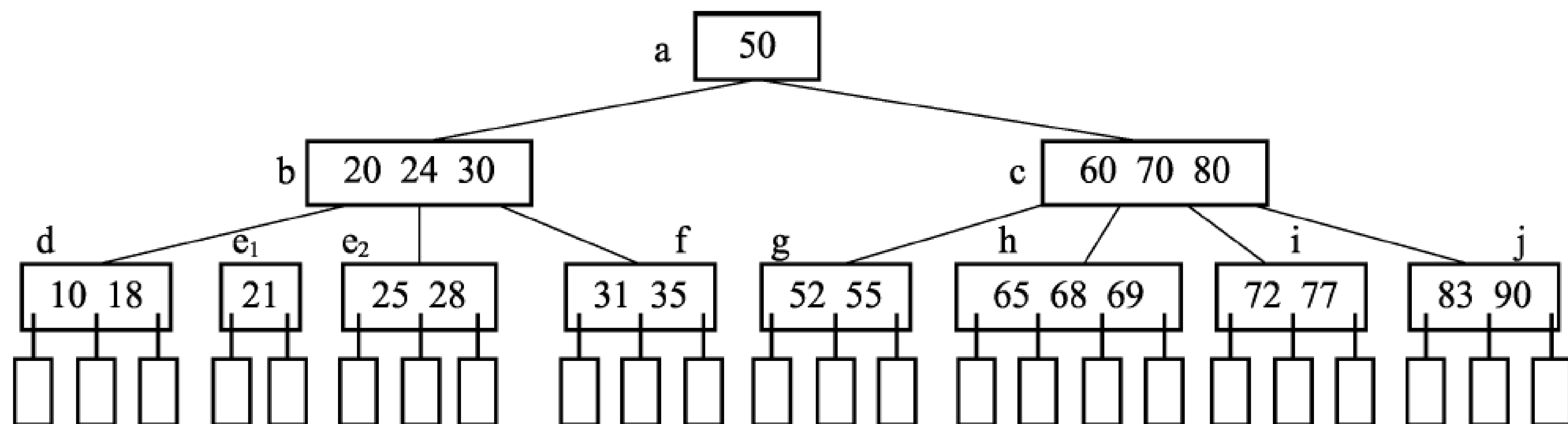


图 5-15　在图 5-14 所示的 4 阶 B 树中插入关键字 25 后的 B 树

函数 Isgrowing(BTreeNode* root,ElemKeyType akey)的功能是：判断在给定的 M 阶 B 树中插入关键字 akey 后，该 B 树的高度是否增加，若增加则返回 TRUE，否则返回 FALSE。其中，root 是指向该 M 阶 B 树根节点的指针。

在函数 Isgrowing 中，首先调用函数 SearchBtree 查找关键字 akey 是否在给定的 M 阶 B 树中，若在则返回 FALSE(表明无须插入关键字 akey，树的高度不会增加)；否则，通过判断节点中关键字的数目考查插入关键字 akey 后该 B 树的高度是否增加。

函数 Isgrowing 的代码如下：

```
bool Isgrowing(BTreeNode* root, ElemKeyType akey)
{
    BTreeNode *t, *f;
    if(!SearchBtree( (4) )){
        t = f;
        while ( (5) ){
```

```
            t = t → parent;
        }
        if(!t)
            return TRUE;
    }
    return FALSE;
}
```

13. 阅读以下函数说明、图和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2005 年 11 月试题四)

【说明】

散列文件的存储单位称为桶(Bucket)。假如一个桶能存放 m 个记录，当桶中已有 m 个同义词(散列函数值相同)的记录时，存放第 m+1 个同义词会发生“溢出”。此时需要将第 m+1 个同义词存放到另一个称为“溢出桶”的桶中。相对地，称存放前 m 个同义词的桶为“基桶”。溢出桶和基桶大小相同，用指针链接。查找指定元素记录时，首先在基桶中查找。若找到，则成功返回；否则沿指针到溢出桶中进行查找。

例如，设散列函数为 Hash(Key)=Key mod 7，记录的关键字序列为 15,14,21,87,96,293,35,24,149,19,63,16,103,77,5,153,145,356,51,68,705,453，建立的散列文件内容如图 5-16 所示。

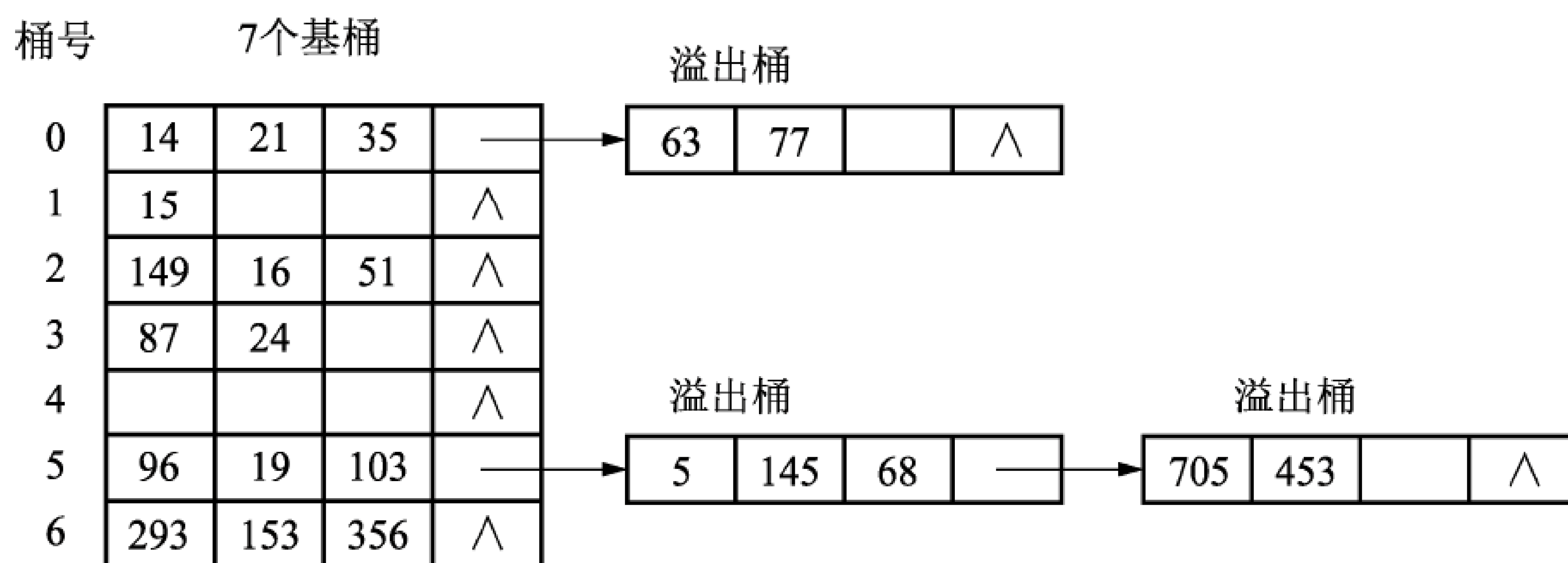


图 5-16 散列文件内容

为简化起见，散列文件的存储单位以内存单元表示。

函数 InsertToHashTable(int NewElemKey)的功能是：若新元素 NewElemKey 正确插入散列文件中，则返回值 1；否则返回值 0。

采用的散列函数为 Hash(NewElemKey)=NewElemKey % P，其中 P 设定基桶数目。

函数中使用的预定义符号如下：

```
#define NULLKEY -1    /*散列桶的空闲单元标识*/
#define P 7           /*散列文件中基桶的数目*/
#define ITEMS 3       /*基桶和溢出桶的容量*/
typedef struct BucketNode{  /*基桶和溢出桶的类型定义*/
    int KeyData[ITEMS];
    struct BucketNode *Link;
}BUCKET;
BUCKET Bucket[P];    /*基桶空间定义*/
```

函数 InsertToHashTable 代码如下：

```
int InsertToHashTable(int NewElemKey){
    /*将元素 NewElemKey 插入散列桶中，若插入成功则返回 0，否则返回-1*/
    /*设插入第一个元素前基桶的所有 KeyData[]、Link 域已分别初始化为 NULLKEY、NULL*/
    int Index;  /*基桶编号*/
    int i, k;
    BUCKET *s, *front, *t;
    (1);
    for(i = 0; i < ITEMS; i++) /* 在基桶查找空闲单元，若找到则将元素存入*/
        if(Bucket[Index].KeyData[i] == NULLKEY){
            Bucket[Index].KeyData[i] = NewElemKey;
            break;
        }
    if((2))return 0;
    /* 若基桶已满，则在溢出桶中查找空闲单元，若找不到则申请新的溢出桶*/
    (3);
    t = Bucket[Index].Link;
    if(t != NULL){  /*有溢出桶*/
        while(t != NULL){
            for(k = 0; k < ITEMS; k++)
                if(t→KeyData[k] == NULLKEY){/* 在溢出桶链表中找到空闲单元*/
                    t→KeyData[k] = NewElemKey;
                    break;
                }/*if*/
            front = t;
            if((4))t = t→Link;
            else break;
        }/*while*/
    }/*if*/
    if((5)){    /* 申请新溢出桶并将元素存入*/
        s = (BUCKET *)malloc(sizeof(BUCKET));
        if(!s)return -1;
        s→Link = NULL;
        for(k = 0; k < ITEMS; k++)
            s→KeyData[k] = NULLKEY;
        s→KeyData[0] = NewElemKey;
        (6);
    }/*if*/
    return 0;
}/*InsertToHashTable*/
```

14. 阅读以下说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2005 年 11 月试题七)

【说明】

在一公文处理系统中，开发者定义了一个公文结构 OfficeDoc，其中定义了公文应该具有的属性。当系统中的文件内容或状态发生变化时，与之相关联的 DocExplorer 结构的值都

需要发生改变。一个 OfficeDoc 结构能够关联一组 DocExplorer 结构。当 OfficeDoc 结构的内容或状态发生变化时，所有与之相关联的 DocExplorer 结构都将被更新，这种应用被称为观察者模式。以下代码采用 C 语言实现，能够正确编译通过。

【C 程序】

```
#include <stdio.h>
#define OBS_MAXNUM 20/*一个 OfficeDoc 变量最多能够关联的 DocExplorer 结构变量的个数*/
typedef void ( (1) )(struct OfficeDoc*, struct DocExplorer*);
struct DocExplorer{
    func update;    /*DocExplorer 结构采用的更新函数*/
    /*其他的结构字段省略*/
};
struct OfficeDoc{
    (2)  myObs[OBS_MAXNUM];
    /*存储所有与 OfficeDoc 相关联的 DocExplorer 结构指针*/
    int index; /*与 OfficeDoc 结构变量相关联的 DocExplorer 结构变量的个数*/
};
void attach(struct OfficeDoc *doc, struct DocExplorer *ob){
    /*关联 DocExplorer 结构 ob 与 OfficeDoc 结构 doc*/
    int loop = 0;
    if(doc->index >= OBS_MAXNUM || ob == NULL)return;
    for(loop = 0; loop < doc->index; loop++)
        if(doc->myObs[loop] == ob)return;
        doc->myObs[doc->index] = ob;
        doc->index++;
}
void detach(struct OfficeDoc *doc, struct DocExplorer *ob){
    /*解除 doc 结构与 ob 结构间的关系*/
    int loop;
    if(ob == NULL)return;
    for(loop = 0; loop < doc->index; loop++){
        if(doc->myObs[loop] == ob){
            if(loop <= doc->index-2)
                doc->myObs[loop] = doc->myObs[ (3) ];
            doc->myObs[doc->index-1] = NULL;
            doc->index--;
            break;
        }
    }
}
void update1(struct OfficeDoc *doc, struct DocExplorer *ob){
    /*更新 ob 结构的值，更新代码省略*/
}
void update2(struct OfficeDoc *doc, struct DocExplorer *ob){
    /*更新 ob 结构的值，更新代码省略*/
}
void notifyObs(struct OfficeDoc *doc){
    /*当 doc 结构的值发生变化时，通知与之关联的所有 DocExplorer 结构变量*/
```

```
        int loop;
        for(loop = 0; loop < doc→index; loop++){
            (doc→myObs[loop]) →update( (4) );
        }
    }
    void main() {
        struct OfficeDoc doc;/*定义一个 OfficeDoc 变量*/
        struct DocExplorer explorer1, explorer2; /*定义两个 DocExplorer 变量*/
        /*初始化与 OfficeDoc 变量相关的 DocExplorer 变量个数为 0*/
        doc.index = 0;
        explorer1.update = update1;     /*设置 explorer1 变量的更新函数*/
        explorer2.update = update2;     /*设置 explorer2 变量的更新函数*/
        attach(&doc, &explorer1);       /*关联 explorer1 与 doc 对象*/
        attach(&doc, &explorer2);       /*关联 explorer2 与 doc 对象*/
        /*其他代码省略*/
         (5) ;    /* 通知与 OfficeDoc 相关的所有 DocExploer 变量*/
        return;
    }
```

15. 阅读下列说明和 C 代码，回答问题 1 至问题 3，将解答写在答题纸的对应栏内。(2016 年 5 月试题四)

【说明】

在一块电路板的上下两端分别有 n 个接线柱。根据电路设计，用$(i,\pi(i))$表示将上端接线柱 i 与下端接线柱$\pi(i)$相连，称其为该电路板上的第 i 条连线。如图 5-17 所示的$\pi(i)$排列为{8,7,4,2,5,1,9,3,10,6}。对于任何 $1\leqslant i<j\leqslant n$，第 i 条连线和第 j 条连线相交的充要条件是$\pi(i)>\pi(j)$。

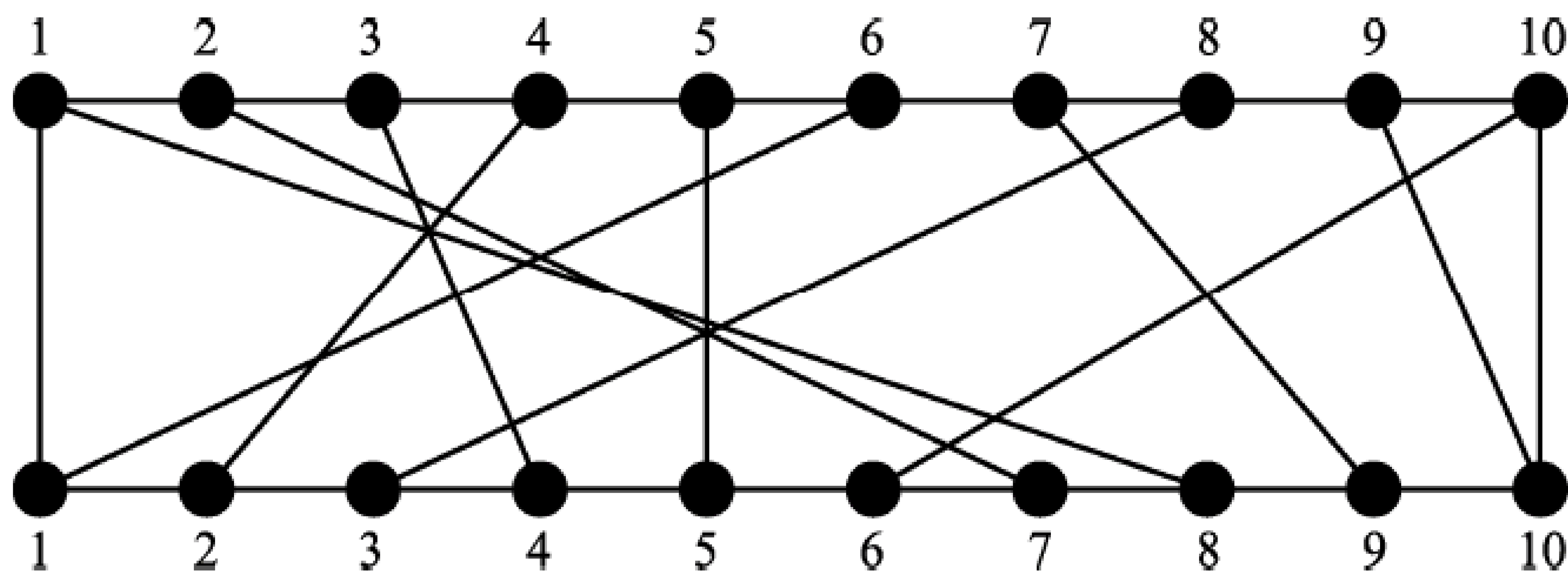


图 5-17　电路布线示意

在制作电路板时，要求将这 n 条连线分布到若干绝缘层上，在同一层上的连线不相交。现在要确定将哪些连线安排在一层上，使得该层上有尽可能多的连线，即确定连线集 $Nets=\{(i,\pi(i)),\ 1\leqslant i\leqslant n\}$的最大不相交子集。

【分析问题】

记 $N(i,j)=\{t|(t,\pi(t))\in Nets, t\leqslant i, \pi(t)\leqslant j\}$。$N(i,j)$ 的最大不相交子集为 $MNS(i,j)$，$size(i,j)=|MNS(i,j)|$。

经分析，该问题具有最优子结构性质。对规模为 n 的电路布线问题，可以构造如下递归式：

(1) 当 i=1 时， $size(1,j)=\begin{cases}0 & j<\pi(1)\\1 & 其他情况\end{cases}$

(2) 当 i>1 时， $size(i,j)=\begin{cases}size(i-1,j) & j<\pi(1)\\\max\{size(i-1,j),size(i-1,\pi(i)-1)+1\} & 其他情况\end{cases}$

【C 代码】

下面是算法的 C 语言实现。

(1) 变量说明

size[i][j]：上下端分别有 i 个和 j 个接线柱的电路板的第一层最大不相交连接数。

pi[i]：π(i)，下标从 1 开始。

(2) C 程序

```
#include "stdlib.h"
#include <stdio.h>
#define   N   10/*问题规模*/
int m=0; /*记录最大连接集合中的接线柱*/
void maxNum(int pi[],int size[N+1][N+1],int n){/*求最大不相交连接数*/
int i, j;
for(j=0; j < pi[1]; j++) size[1][j] = 0; /*当 j<π(1)时*/
for(j=pi[1];j<=n;j++)(1) ; /*当 j>=π(1)时*/
for(i=2; i < n; i++) {
for(j=0; j < pi[i]; j++)(2) ; /*当 j<pi[i]时*/
for(j=pi[i];j<=n;j++){/*当 j>=c[i]时，考虑两种情况*/
 size[i][j]=size[i-1][j]>=size[i-1][pi[i]-1]+1 ?size[i-1][j]:
size[i-1][pi[i]-1]+1;
}
}
/*最大连接数*/
size[n][n]=size[n-1][n]>=size[n-1][pi[n]-1]+1 ?
size[n-1][n]:size[n-1][pi[n]-1]+1;
}
/*构造最大不相交连接集合，net[i]表示最大不相交子集中第 i 条连线的上端接线柱的序号*/
void constructSet(int pi[],int size[N+1][N+1],int n,int net[n]){
int i,j=n;
m=0;
for(i=n ; i>1 ; i--){/*从后往前*/
if(size[i][j]!=size[i-1][j]){/*(i,pi[i])是最大不相交子集的一条连线*/
(3);/*将 i 记录到数组 net 中，连接线数自增 1*/
j= pi[i]-1;/*更新扩展连线柱区间*/
}
}
if(j>=pi[1])net[m++]=1;/*当 i=1 时*/
}
```

【问题 1】

根据以上说明和 C 代码，填充 C 代码中的空(1)~(3)。

【问题 2】

根据题干说明和以上 C 代码，算法采用了__(4)__设计策略。

函数 maxNum 和 constructSet 的时间复杂度分别为__(5)__和__(6)__(用 O 表示)。

【问题 3】

若连接排列为{8,7,4,2,5,1,9,3,10,6}，即如图 5-17 所示，则最大不相交连接数为__(7)__，包含的连线为__(8)__(用(i,π(i))的形式给出)。

5.1.4 同步练习参考答案

1.

【问题 1】

(1) k←0。 (2) cw←cw+w[k]。 (3) Y[k]←X[k]。 (4) k←k+1。

【问题 2】

(5) 2。 (6) 18。 (7) 15。 (8) 4。

2.

(1) InitStack(&station)。

(2) !IsEmpty (station)。

(3) state[i]<Top(station)。

(4) Top(station)。

(5) j。

3.

【问题 1】

(1) k=1 to n。 (2) $d_{ij}^{(k)}=d_{ij}^{(k-1)}$。 (3) $d_{ij}^{(k)}=d_{ik}^{(k-1)}+d_{kj}^{(k-1)}$。

(4) SP[i]+=$d_{ij}^{(n)}$。 (5) min_v=1。 (6) min_v。

【问题 2】

(7) $O(n^3)$。

4.

(1) ptr!=NULL。 (2) q→link=stacktop。 (3) ptr=ptr→lchild。

(4) stacktop=stacktop→link。 (5) q→rchild。

5.

【问题1】

(1) nv[i-1][j] >= nv[i-1][j-p[i]] + v[i]。

(2) nv[i][j] = = nv[i-1][j]。

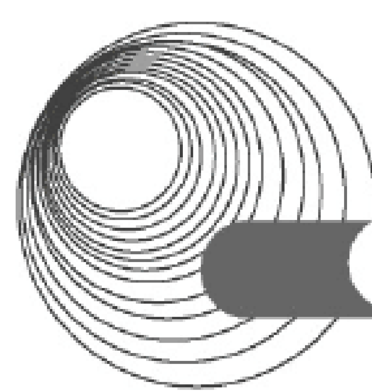
(3) j = j-p[i]。

【问题 2】

(4) m_2,m_3,m_4。 (5) 605。

【问题 3】

(6) O(n×M)。



6.

(1) pa=*LA。 (2) pb&&pb→elem!=pa→elem。 (3) pb。

(4) pa→next 或(*pa).next 或*pa.next。 (5) pre→next 或(*pre).next 或*pre.next。

(6) pre=pa。

7.

【问题 1】

(1) A[i+1]。 (2) A[r]。

注：(1)和(2)的内容可互换。

(3) (i+1)或++i。

{或者(1) A[++i]。 (2) A[r]或 A[q]。 (3) i (其中(1)和(2)的内容可互换)}。

【问题 2】

(4) $O(n\log_2 n)$或 O(nlgn)。 (5) $O(n\log_2 n)$或 O(nlgn)。 (6) $O(n^2)$。 (7) 最坏。

【问题 3】

(8) A[i]。 (9) A[r]。

注：(8)和(9)可互换。

(10) 否。

8.

(1) S= =NULL || S→pTop = =NULL。

(2) S→pTop→data。

(3) newNode。

(4) S→pTop→next。

(5) 24 4。

9.

(1) this→center。 (2) this→center→common。

(3) p_circle→center。 (4) va_start。

(5) C(P(20,40),10)。

10.

(1) root。 (2) root→childptr[0]。 (3) root→childptr[i]。

(4) placeBoosters(p)。 (5) degradation。

11.

(1) EnQueue(&tempQ,root)。 (2) brotherptr=brotherptr→nextbrother。

(3) !IsEmpty(tempQ)。 (4) DeQueue(&tempQ,&ptr)。 (5) !ptr→firstchild。

(6) EnQueue(&tempQ,ptr→firstchild)。 (7) brotherptr=brotherptr→nextbrother。

12.

(1) p→numkeys 或其等价形式。

(2) p→K[mid]>akey 或其等价形式。

(3) p→A[hi]或 p→A[lw−1]或其等价形式。

(4) root，akey，&f。

(5) t&&t→numkeys==M−1 或其等价形式。

13.

(1) Index = NewElemKey%P 或 Index = Hash(NewElemKey)。

(2) i < ITEMS。

(3) front = &Bucket[Index]或 front=Bucket+Index。

(4) k == ITEMS 或 k >= ITEMS。

(5) t == NULL。

(6) front → Link = s。

14.

(1) *func。　(2) struct DocExplorer*。　(3) doc → index-1。

(4) doc, doc → myObs[loop]。　(5) notifyObs(&doc)。

15.

【问题 1】

(1) size[i][j]=1。

(2) size[i][j]=size[i-1][j]。

(3) net[m++]=i;。

【问题 2】

(4) 动态规划算法。

(5) $O(n^2)$。

(6) O(n)。

【问题 3】

(7) (9，π(9))，(7，π(7))。

(8) (5，π(5))，(3，π(3))。

5.2 本章小结

本章知识点在 2009 年的新大纲中改动不大。

每个考生都知道，下午考试科目分为必做和选做两部分。根据近几年软件设计师水平考试试题分布情况来看，算法设计题目有时放在必做题目中，有时放在选做题目中，所以还是建议考生掌握相关的内容。其占的分数也是一定的，几个程序填空，总共 15 分。

算法设计的考试主要围绕几种重要的数据结构(如栈、队列和树等)和几种典型的算法问题(背包问题、最短路径问题、几种排序算法)出题，考题有一定难度，但只要考生掌握这几个重要数据结构和几种典型算法的基本思想，结合题目中对算法的描述以及已经给出的程序，把程序或者伪代码补充完整也是不难的。

第 6 章　面向对象程序设计

大纲要求：

掌握 C++、Java 中任一种程序设计语言，以便能指导程序员进行编程和测试，并进行必要的优化。

6.1　C++基础知识

6.1.1　考点辅导

6.1.1.1　C++概述

C++是由 C 发展成为的以面向对象为主要特征的语言。作为 C 语言的超集，C++继承了 C 的所有优点，且对数据类型做了扩充，使得编译系统可以检查出更多类型错误。C++支持面向对象程序设计，通过类和对象的概念把数据和对数据的操作封装在一起，通过派生、继承、重载和多态性等特征实现了软件重用和程序自动生成，使得大型复杂软件的构造和维护变得更加有效和容易。此外，C++在一致性(Consistency)检查机制方面也作了加强，提高了软件开发的效率和质量。

1. 词法单位

1) 关键字

数据类型说明符与修饰符：bool、char、wchar_t、class、const、double、enum、float、int、long、short、signed、struct、union、unsigned、void、volatile。

存储类型说明符：auto、extern、inline、register、static。

访问说明符：friend、private、protected、public。

其他说明符：asm、operator、template、this、typedef、virtual。

语句与标号：break、case、catch、continue、default、do、else、for、goto、if、return、switch、throw、try、while。

运算符及逻辑值：delete、false、new、sizeof、true。

2) 标识符

合法标识符由字母或下画线开始，由字母、数字、下画线组成，不能是 C++关键字。其有效长度为 1~31 个字符，长度超过 31 个字符者只识别前 31 个字符，VC++标识符长度为 1~247 个字符。

2. 数据类型

基本数据类型是 C++内部预先定义的数据类型，非基本数据类型是用户自己定义的数

据类型。

1) 基本数据类型

基本数据类型包括整型 int、字符型 char、逻辑型 bool、无值型 void、实型 float、双精度型 double，与 C 语言没什么差别。

2) 非基本数据类型

非基本数据类型包括数组 type []、指针 type *、结构 struct、联合 union、枚举 enum 和类 class，比 C 语言增加了类 class 数据类型。

3) new 和 delete

(1) 分配内存。

在 C 语言中：char *name = (char *)malloc(Length + 1);。

使用 new：char *name = new char[Length + 1];。

(2) 释放内存。

在 C 语言中：free(name);。

使用 delete：delete [] name;。

3. 运算符及其优先级

1) 运算符

- 算术运算符：+、−、*、/、%，自增、自减运算符++、−−。
- 关系运算符：>、<、==、>=、<= 和!=。
- 逻辑运算符：&&、||、!。
- 位操作运算符：&、|、~、^、<<、>>。
- 赋值运算符。
    - 简单赋值：=。
    - 复合算术赋值：+=、−=、*=、/=、%=。
    - 复合位运算赋值：&=、|=、^=、>>=、<<=。
- 条件运算符： c ? a : b。
- 逗号运算符：用于把若干表达式组合成一个表达式(,)。
- 指针运算符：用于取内容(*)和取地址(&)两种运算。
- 求字节数运算符：sizeof。
- 特殊运算符：括号()、下标[]、成员(->、.)等。

2) 优先级

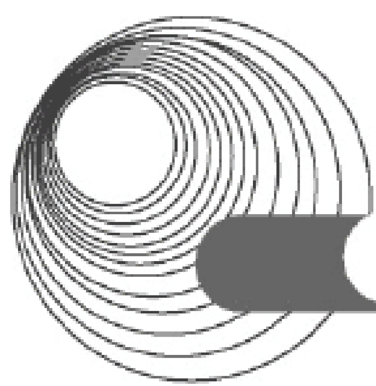
运算首先按优先级进行，如果运算对象两侧的运算符具有相同的优先级，则按照结合性处理。

优先级由高到低依次为：括号；++、−−、sizeof、!(右结合)；*、/、%；+、−；<<、>>；>、<、<=、>=；==、!=；位运算；&&；||；? : (右结合)；赋值运算(右结合)；逗号运算。

4. 算法的基本控制结构

1) 顺序

顺序结构依次执行各语句。



2) 选择

- 条件语句：if_else。
- 开关语句：switch。

3) 循环

- for 循环。
- while 循环。
- do…while 循环。

5. 输入/输出

在 C++中把数据的 I/O 称为数据流，并提供了强大的“流”处理功能，以控制数据从一个位置流向另一个位置。

这里输入/输出是相对内存来说的。当数据从内存流向屏幕、打印机或硬盘时称为输出；当数据从键盘、硬盘流向内存时称为输入。

在 C++中用 istream 类和 ostream 类的派生类 iostream 控制输入/输出，用两个对象 cin 和 cout 实现标准的输入/输出，并提供了输入/输出操作符。

- <<：插入操作符，其作用是向 cout 流中插入字符。
- >>：抽取操作符，其作用是从 cin 流中提取字符。

其格式如下。

- cin >> <表达式> >> <表达式>...：读取键盘输入的数据，并由后面的变量保存下来。
- cout << <表达式> << <表达式>...：将结果显示在屏幕上。

另外，输出时常用到 endl，它是转义字符，意思是当前行结束，所以下一次输出会显示在下一行上。

6.1.1.2 函数

C++同 C 一样，也是函数驱动，程序入口也是 main 函数。有关虚函数的概念详见 6.1.1.5 节。

1. 函数的定义和调用

1) 函数的定义

函数的定义格式为：

```
[<数据类型>] <函数名>([<参数类型 1> <形式参数 1>][,<参数类型 2><形式参数 2>,…])
{
<函数体>
}
```

C++允许有默认值，定义时指明默认变量的默认值。值得注意的是，若形参有多个时，默认变量必须在最后声明。如 void power(double x, int n=2)表示 n 的默认值为 2。

2) 函数的调用

- 无参函数的调用格式为：函数名()。
- 有参函数的调用格式为：函数名(实际参数表)。

对于有默认形参的函数，调用形式更为灵活。如前面的例子，调用 power(x)时相当于 power(x, 2)。

3) 参数传递

C++中函数的参数传递与 C 语言基本类似。另外，C++增加了引用传值，传递的是地址，相当于别名。如定义函数 void exchange(double &x1, double &x2)，调用时 exchange(x1, x2)，结果与 C 语言中的指针传值是一致的：定义 void exchange(double *x1, double *x2)，调用 exchange(&x1, &x2)。

2. 内联函数

内联函数不是在调用时发生控制转移，而是在编译时将函数体嵌入在每一个调用语句处。

内联函数的定义格式如下：

```
inline 类型说明符 函数名(形参表)
```

使用内联函数时应注意以下几点。

(1) 内联函数一般不包含循环语句和 switch 语句。

(2) 内联函数的定义必须出现在第一次调用之前。

(3) 对内联函数不能进行异常接口声明。

3. 重载

函数名相同而形参不同(类型或个数)的两个函数就构成了重载。仅返回类型不同的两个函数不能构成重载。

4. 函数模板

模板是为了替代容易出错的宏而提出的。模板分为函数模板和类模板，这里先讨论函数模板。函数模板其实就是定义了一系列仅数据类型不同的函数。

函数模板的定义格式如下：

```
template <数据类型列表> 返回值 函数名(形参列表)
```

函数和模板的匹配顺序如下。

(1) 先找一个参数完全匹配的函数。

(2) 再找一个参数完全匹配的模板。

(3) 再找一个参数经过自动转换后能够匹配的函数。

(4) 都找不到，则报错。

6.1.1.3　类与对象

1. 类和对象

1) 类的概念

类是数据以及用于操纵该数据的方法(函数)的集合，是逻辑上相关函数与数据的封装。它是对所要处理问题的抽象描述，它把数据(事物的属性)和函数(事物的行为/操作)封装为一个整体。

2) 类的定义格式

类的定义格式如下：

```
class 类名{
private:
```

```
    //私有数据和函数
public:
    //公有数据和函数
protected:
    //保护数据和函数
};
```

其中，private、public、protected 称为访问权限控制关键字，其作用是限制“可见性”。在类的定义中，以上 3 种关键字出现的次数和先后次序都没有限制，默认被认为是 private。这 3 种关键字对应的访问可见性如下。

- private：私有成员，只能在成员函数内访问。
- public：公有成员，可以在对象的外部访问。
- protected：保护成员，只能由对象内部或其派生类对象访问。

面向对象的思想是对对象的属性(成员变量)进行操作，应该通过对象的方法(成员函数)来进行，对象的方法是对象和外部的接口。将类的成员变量声明为 private，能保证对该成员的操作都是通过类的方法来进行的。这样可以避免出错(如对成员变量不恰当地赋值)，也便于修改程序。在修改类的定义时，只要修改类的成员函数就可以了，不需要修改使用该类成员的代码。如果某些成员函数只被其他成员函数调用，不作为对象的界面，那么也可以将它声明为 private。

成员函数定义通常在类的说明之后进行，其格式如下：

```
返回值类型 类名::函数名(参数表)
{
    //函数体
}
```

其中，运算符“::”称为作用域解析运算符，它指出该函数是属于哪一个类的成员函数。当然也可以在类的定义中直接定义函数。

3) 对象

对象即类的实例(Instance)。

创建类的对象可以有两种常用方法。

(1) 第一种是直接定义类的对象。如 CGoods Car 这个定义创建了 CGoods 类的一个对象 Car，同时为它分配了属于它自己的存储块，用来存放数据和对这些数据实施操作的成员函数(代码)。与变量定义一样，一个对象只在定义它的域中有效。第一种方法通过“对象名.成员名”方式访问对象成员。

(2) 第二种是采用动态创建类的对象的方法，当然变量也同样可动态创建。所谓动态，是指在程序运行时建立对象。而第一种方法是在编译时(程序运行前)建立。第二种方法通过对象指针访问对象成员。

2. 构造函数和析构函数

1) 构造函数

构造函数是特殊的成员函数，其特征如下。

(1) 是成员函数的一种，名字与类名相同，可以有参数，不能有返回值(void 也不行)。

(2) 其作用是对对象进行初始化，如给成员变量赋初值。

(3) 如果定义类时没写构造函数，则编译器生成一个默认的无参数的构造函数。默认构造函数无参数，什么也不做。默认的构造函数也可以由程序员自己来编写，只要构造函数是无参的或者只要各参数均有默认值，C++编译器都认为是默认的构造函数，并且默认的构造函数只能有一个。

(4) 如果定义了构造函数，则编译器不生成默认的无参数的构造函数。

(5) 对象生成时，构造函数自动被调用。

(6) 一个类可以有多个构造函数，它们的参数个数或类型不同，构成函数重载。

构造函数一般声明为 public，当然有时为了特殊需要也可定义为 private。

2) 复制构造函数

同一个类的对象在内存中有完全相同的结构，如果作为一个整体进行复制是完全可行的。这个复制过程只需要复制数据成员，而函数成员是共用的(只有一份副本)。

在建立对象时可用同一类的另一个对象来初始化该对象，这时所用的构造函数称为复制初始化构造函数(Copy Constructor)。形如 X::X(X&)，只有一个参数——同类(Class)的对象，采用的是引用的方式。不允许有形如 X::X(X)的构造函数，如果把一个真实的类对象作为参数传递到复制构造函数，会引起无穷递归。如果没有定义，那么编译器生成默认复制构造函数。如果定义了自己的复制构造函数，则默认的复制构造函数不存在。

复制构造函数在 3 种情况下被调用。

(1) 当用一个对象去初始化同类的另一个对象时。

```
Complex c2(c1);
Complex c2 = c1;
```

(2) 如果某函数有一个形参是类 A 的对象，那么该函数被调用时，类 A 的复制构造函数将被调用。

```
void f(A a){
    a.x = 1;
};
A aObj;
f(aObj);
//A 的复制构造函数被调用，生成形参，在内存新建立一个局部对象，并把实参复制到新的对象中
```

(3) 如果函数的返回值是类 A 的对象，则函数返回时，A 的复制构造函数被调用。理由也是要建立一个临时对象，再返回调用者。

```
A f(){
    A a;
    return a; // 调用 A(a);
}
int main( ) {
    A b; b = f();
    return 0;
}
```

为什么不直接用要返回的局部对象呢？因为局部对象在离开建立它的函数时就消亡

了，不可能在返回调用函数后继续生存。所以编译系统会在调用函数的表达式中创建一个无名临时对象，该临时对象的生存周期只在函数调用处的表达式中。所谓 return 对象，实际上是调用复制构造函数把该对象的值复制到临时对象。

3) 析构函数

当一个对象定义时，C++自动调用构造函数建立该对象并进行初始化，那么当一个对象的生命周期结束时，C++也会自动调用一个函数注销该对象并进行善后工作，这个特殊的成员函数即析构函数(Destructor)。

(1) 析构函数名也与类名相同，但在前面加上字符“~”，如~CGoods()。

(2) 析构函数无函数返回类型，在这方面与构造函数是一样的，但析构函数不带任何参数。

(3) 一个类有一个也只有一个析构函数，这与构造函数不同。析构函数可以缺省。

(4) 对象注销时，系统自动调用析构函数。

4) 一些补充说明

对于不同作用域的对象类型，构造函数和析构函数的调用如下。

(1) 对于全局定义的对象，当程序进入入口函数 main 之前对象就已经定义，这时要调用构造函数。整个程序结束时，调用析构函数。

(2) 对于局部定义的对象，每当程序控制流到达该对象定义处时，调用构造函数。当程序控制流走出该局部域时，调用析构函数。

(3) 对于静态局部定义的对象，在程序控制流首次到达该对象定义处时，调用构造函数。当整个程序结束时，调用析构函数。

5) 成员对象

有成员对象的类称为封闭(Enclosing)类。对成员对象初始化，必须调用该成员对象的构造函数来实现。先调用所有对象成员的构造函数，然后才调用封闭类的构造函数。对象成员的构造函数调用次序和对象成员在类中的说明次序一致，与它们在成员初始化列表中出现的次序无关。当封闭类的对象消亡时，先调用封闭类的析构函数，然后再调用成员对象的析构函数，次序和构造函数的调用次序相反。

例如：

```
class base1 {
private:
    int i;
public:
    base1(){ i=0; }
    base1(int n) { i=n; }
};
class Big {
private:
    int n;
    base1 b1;
public:
    Big() : b1(1){ }
    Big(int n) : b1(n){ }
};
```

6.1.1.4　继承与派生

1) 基本概念

继承(Inheritance)机制是面向对象程序设计使代码可以复用的最重要手段，它允许程序员在保持原有类特性的基础上，调整部分成员的特性，也可以增加一些新成员。

通过继承，能够以已有的类为基础定义新的类，使新的类具有已有类的全部特点和功能，新的类还能添加自己的特点和功能，或修改老的类的特点和功能。已有的类(被继承的类)称为基类或父类，新的类(继承的类)称为派生类或子类。

具体地，派生类拥有基类的全部成员变量和成员函数，而且还能添加新的成员变量和成员函数，也可以重新定义从基类继承的成员变量和成员函数，即吸收基类成员、改造基类成员、添加新的成员。继承和派生机制大大地提高了软件的可重用性和可扩充性。

2) 访问控制

C++提供了 3 种继承方式，也是用 public、protected、private 三个关键字标识，一般采用公有继承 public。3 种继承方式的具体意义如表 6-1 所示。

表 6-1　3 种继承方式的具体意义

派生方式	基类中的访问限定	在派生类中对基类成员的访问限定	在派生类对象外访问派生类对象的基类成员
公有派生 public	public	public	可直接访问
	protected	protected	不可直接访问
	private	不可访问	不可直接访问
保护派生 protected	public	protected	不可直接访问
	protected	protected	不可直接访问
	private	不可访问	不可直接访问
私有派生 private	public	private	不可直接访问
	protected	private	不可直接访问
	private	不可访问	不可直接访问

3) 赋值兼容规则

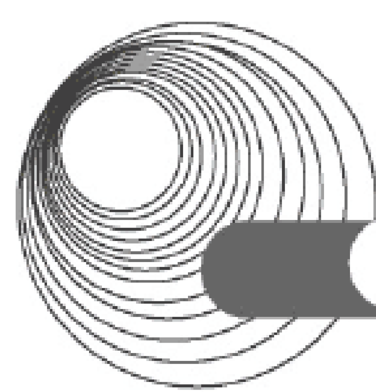
在需要基类对象的地方可以使用公有派生类来替代，派生类对象能自动地当作其基类对象来使用，但基类对象不能当作其派生类对象来使用。这正体现了“派生类对象是一个基类对象”。

具体使用情况如下。

- 派生类对象可以赋值给基类对象：b = d。
- 派生类对象可以初始化基类引用：base &br = d。
- 派生类对象的地址可以赋值给基类指针：base *pb = &d。

4) 重置(覆盖)

派生类可以定义一个和基类成员同名的成员，这称为覆盖。派生类成员将覆盖所有基类的同名成员，默认的情况是引用派生类的成员。若想访问基类同名成员，需要通过域作用符“::”——基类名::数据成员名、基类名::函数成员名(参数表)。



6.1.1.5 多态

多态性(Polymorphism)同继承性一样，是面向对象程序设计的标志性特征，是一个考查重点。

多态性是考虑在不同层次的类中以及在同一类中，同名的成员函数之间的关系问题。函数的重载和运算符的重载都属于多态性中的编译时的多态性。运行时的多态性是以虚基类为基础的多态性。

1) 多态的定义

多态是指同样的消息被不同类型的对象接受时导致不同的行为(不同的实现或调用了不同的函数)。所谓消息，是由“类::方法”(功能)和“方法的实参”(消息数据)共同组成的。

产生多态性的原因是：不同的对象在处理同样的消息时，使用的方法实现(成员函数的函数体)不同。“多态性”是与“类的派生和继承”联系在一起的，是基类中所定义方法的“多态性”。对于在派生类中新增加的方法，是没有多态性的。

2) 分类

(1) 重载多态：成员函数(运算符)重载。

(2) 强制多态：强制类型转换。把一个变量的类型变换成另一种类型，以符合一个函数或者操作的要求。例如，加法运算符在执行浮点数和整数的相加时，首先把整数转换成浮点数，然后再相加。

(3) 包含多态：主要通过虚函数来实现。强调不同类中的同名成员函数的多态行为。

(4) 参数多态：可通过函数模板和类模板来实现。

在 C++中有两种多态性。

(1) 编译时的多态性：通过函数的重载和运算符的重载来实现。

(2) 运行时的多态性：是指在程序执行前，无法根据函数名和参数来确定该调用哪一个函数，必须在程序执行过程中，根据执行的具体情况来动态地确定。这种多态性是通过类继承关系和虚函数(Virtual Function)来实现的。

3) 虚函数

虚函数是前面有 virtual 关键字的类的成员函数，定义虚函数的格式如下：

```
virtual 返回类型 函数名(参数表);
```

注意：virtual 关键字只用在类定义里的函数声明中，写函数体时不用。

另外，如果基类中的函数不是虚函数，即没有 virtual 关键字，即使派生类中写了 virtual 也没有用，不能实现多态。

使用虚函数时，需要注意以下几点。

(1) 派生类中定义虚函数除必须与基类中的虚函数同名外，还必须同参数表、同返回类型。如基类中返回基类指针，派生类中返回派生类指针是允许的。

(2) 只有类的成员函数才能说明为虚函数。

(3) 静态成员函数不能作为虚函数。

(4) 实现动态多态性时，必须使用基类指针或引用，使该指针指向不同派生类的对象，并指向虚函数。

(5) 内联函数不能作为虚函数。

(6) 析构函数可定义为虚函数，构造函数不能为虚函数。在基类中及其派生类中都有动态分配的内存空间时，必须把析构函数定义为虚函数，实现撤销对象时的多态性。

4) 纯虚函数和抽象类

(1) 纯虚函数(Pure Virtual Function)：指被标明为不具体实现的虚拟成员函数。定义纯虚函数的一般格式为：

```
virtual 返回类型 函数名(参数表) = 0;
```

例如：

```
class A {
public:
int a;
virtual void Print() = 0 ; //纯虚函数
};
```

定义纯虚函数必须注意以下几点。

- 定义纯虚函数时，不能定义虚函数的实现部分。
- “= 0”本质上是将指向函数体的指针定义为 NULL。
- 在派生类中必须有重新定义的纯虚函数的函数体，这样的派生类才能用来定义对象。

(2) 抽象类：包含纯虚函数的类。

抽象类只能作为基类来派生新类使用，不能创建抽象类的对象，可声明一个抽象类的指针和引用。

在抽象类的成员函数内可以调用纯虚函数，但是在构造函数或析构函数内部不能调用纯虚函数。因为在构造函数或析构函数内部调用虚函数采用的是静态联编，即编译时就要生成调用该函数的指令，而纯虚函数是没有代码的，所以这样的调用指令无法生成，因此编译会报错。在普通成员函数内可以调用纯虚函数，尽管纯虚函数是没有代码的，但是此时是动态联编，编译时不需要生成调用该函数的指令，所以编译可以通过。在运行时决定到底调用的是自己还是派生类的函数，因为自己是个抽象类，不可能生成对象，所以不可能调用自己的这个纯虚函数。

5) 虚析构函数

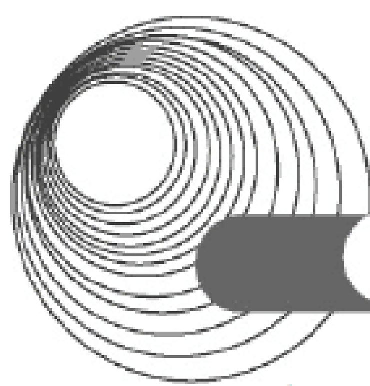
只要基类的析构函数是虚函数，那么派生类的析构函数不论是否使用 virtual 关键字，不论是自己定义的还是编译器默认生成的，都自动成为虚函数。

一个类的构造函数会在执行自己代码之前，依派生顺序自动调用它的所有直接基类的构造函数；一个类的析构函数也会在执行完自己的代码之后，以与构造函数调用次序相反的顺序自动调用其所有直接基类的析构函数。一般来说，一个类如果定义了虚函数，则应该将析构函数也定义成虚函数。

6.1.1.6　异常处理

这里所讲的异常(Exception)是程序可能检测到的运行时不正常的情况，如存储空间耗尽、数组越界、被 0 除等，可以预见可能发生在什么地方，但是无法确知怎样发生和何时发生。

try{…}catch{…}搭配可实现异常的捕获处理。通常将有可能产生异常的程序块放在 try



中，catch 捕获产生的异常并进行处理或重新抛出异常。当程序没有异常时，忽略 catch 块，程序正常执行。

C++中，异常一般通过类来实现，throw 表达式通过调用异常类的构造函数创建一个临时对象，然后把这个临时对象复制到一个被称为异常对象(Exception Object)的存储区中，它保证会持续到异常被处理完。这样抛出异常的一般格式如下：

```
throw new 异常类型
```

C++标准库提供了一个异常类层次结构，用来报告 C++标准库中的函数执行期间遇到的程序不正常情况。这些异常类也可以被用在用户编写的程序中，或被进一步派生来描述程序中的异常。

C++标准库中的异常层次的根类被称为 exception，定义在库的头文件<exception>中，它是 C++标准库函数抛出的所有异常类的基类。

exception 类的接口如下：

```
namespace std{ //注意在名字空间域 std 中
    class exception{
    public:
    exception() throw();//默认构造函数
    exception(const exception &) throw();//复制构造函数
    exception &operator=(const exception&) throw();//复制赋值操作符
    virtual ~exception() throw();//析构函数
    virtual const char*what() const throw();//返回一个 C 风格的字符串
    };
}
```

6.1.2 典型例题分析

例 1 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2013 年 5 月试题五)

【说明】(15 分)

现要求实现一个能够自动生成求职简历的程序，简历的基本内容包括求职者的姓名、性别、年龄及工作经历。希望每份简历中的工作经历有所不同，并尽量减少程序中的重复代码。

现采用原型模式(Prototype)来实现上述要求，得到如图 6-1 所示的类图。

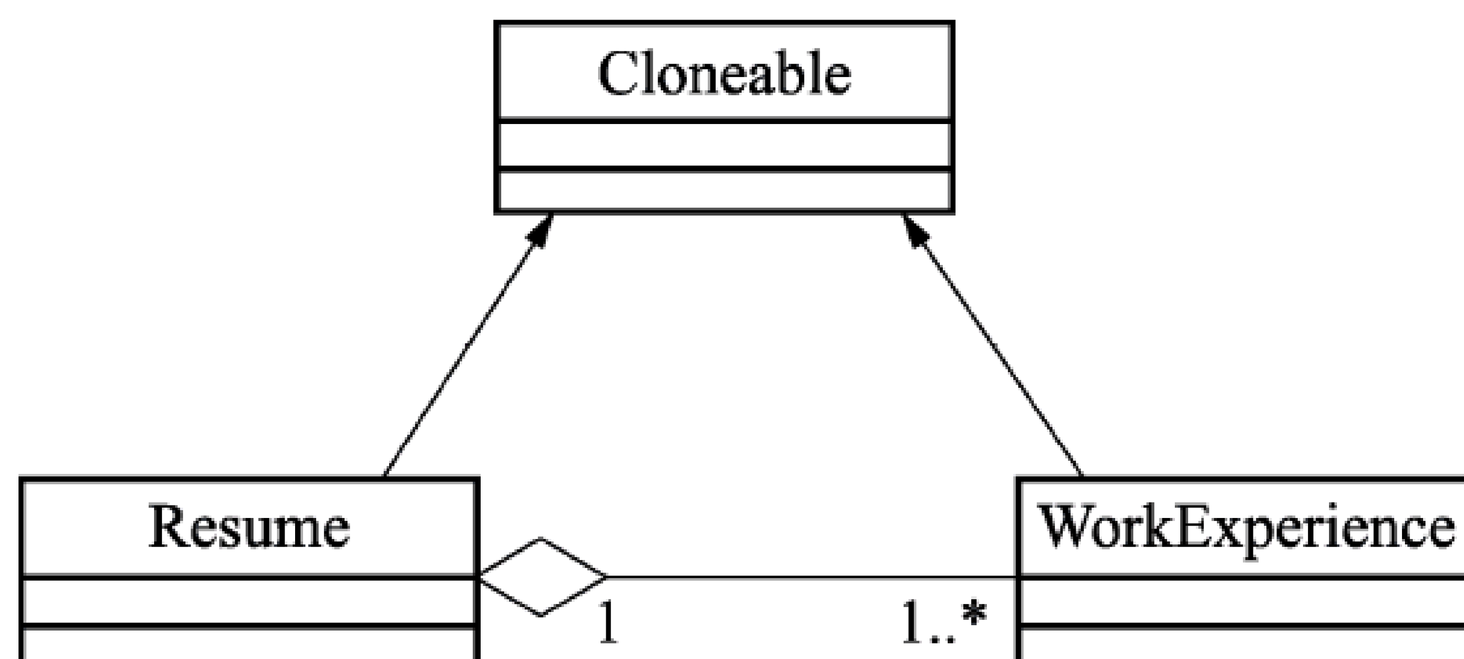
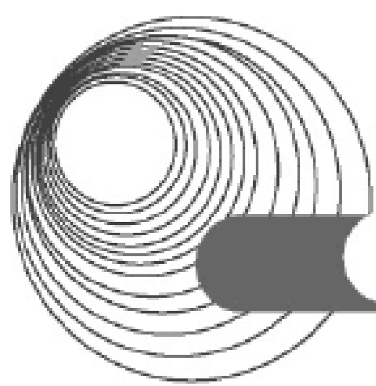


图 6-1　例 1 类图

【C++代码】

```
#include <string>
using namespace std;
class Cloneable{
public:
     (1) ;
};
Class WorkExperience:public Cloneable{    //工作经历
private:
     string  workDate;
     string  company;
public:
    Cloneable*Clone(){
        (2) ;
        Obj->workDate=this->workDate;
        Obj->company=this->company;
        Return obj;
  }
    //其余代码省略
};
class Resume:public Cloneable{  //简历
private:
   string  name;  string sex;  string  age;
   WorkExperience* work;
   Resume(WorkExperience*work){
      This->work= (3) ;
   }
public:
   Resume(string name){  /* 实现省略*/    }
      void  SetPersonalInfo(string sex,string age){     /* 实现省略*/     }
      void setWorkExperience(string  workDate,string company)
      {  /* 实现省略*/   }
      Cloneable*Clone(){
          (4) ;
          Obj->name=this->name;
          Obj->sex=this->sex;
          Obj->age=this->age;
          return obj;
   }
};
int main(){
   Resume *a=new Resume("张三");
   A->SetPersonalInfo("男","29");
   A->SetWorkExperience("1998~2000","XXX 公司");
   Resume*b= (5) ;
   B->SetWorkExperience("2001~2006","YYY 公司");
   return 0;
}
```



解析：

本题考查原型模式的概念及应用。原型模型是一种对象创建模型，用原型实体指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模型允许一个对象再创建另一个可定制的对象，无须知道任何创建的细节。

原型模式其实就是常说的“虚拟构造函数”的一个实现，C++的实现机制中并不支持这个特性，但是通过不同派生类实现的 Clone 接口函数，可以完成与“虚拟构造函数”同样的效果。

本题中声明一个虚拟基类，所有的原型都从这个基类继承，空(1)处所代表的就是这个基类中的纯虚函数，需要供继承者自行实现，因此空(1)处应填入 virtual Cloneable * Clone() = 0，声明一个抽象基类，并定义 Clone()函数为纯虚函数。然后根据基类实例化各个子类，并且实现复制构造函数，并实现 Clone()函数，由此可知空(2)处应填入 WorkExperience *obj，空(3)处应填入 Work，空(4)处应填入 Resume *obj。在 main 函数中实现 Resume *b 对*a 的复制，故根据 C++语法，空(5)处应填入 a->Clone()。

答案：

(1) virtual Cloneable * Clone() = 0。　　(2) WorkExperience *obj。

(3) Work。　　(4) Resume *obj。　　(5) a->Clone()。

例 2　阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2012 年 11 月试题五)

【说明】(共 15 分)

现欲开发一个软件系统，要求能够同时支持多种不同的数据库，为此采用抽象工厂模式设计该系统。以 SQL Server 和 Access 两种数据库以及系统中的数据库表 Department 为例，其类图如图 6-2 所示。

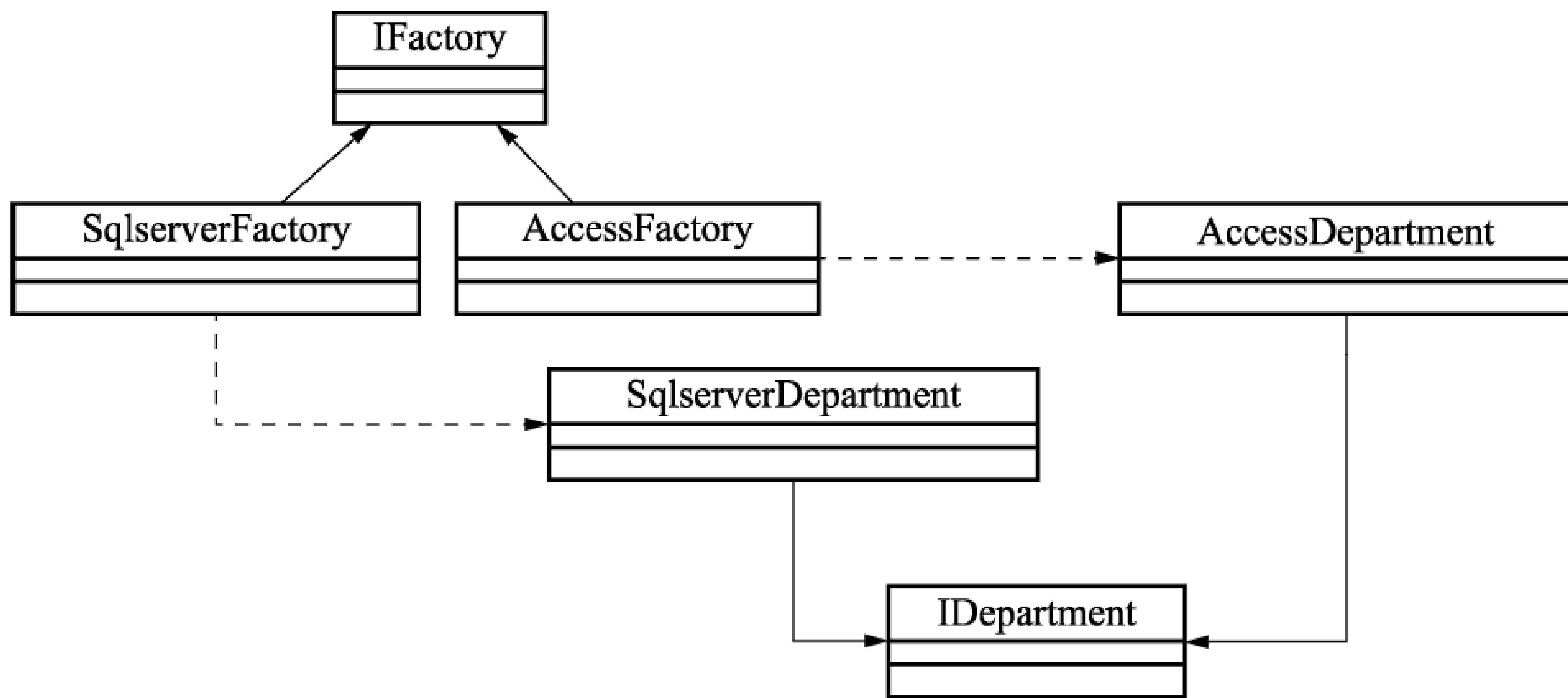


图 6-2　例 2 类图

【C++代码】

```
#include <iostream>
using namespace std;
class Department{/*代码省略*/};
class IDepartment{
```

```
public:
    ___(1)___=0;
    ___(2)___=0;
};
class SqlserverDepartment:___(3)___{
public:
    void Insert(Department* department){
        cout<<"Insert a record into Department in SQL Server!\n";
        //其余代码省略
    }
    Department GetDepartment(int id){
        /*代码省略*/
    }
};
class AccessDepartment:___(4)___{
public:
    void Insert(Department* department){
        cout<<"Insert a record into Department in ACCESS!\n";
        //其余代码省略
    }
    Department GetDepartment(int id){
        /*代码省略*/
    }
};
___(5)___{
public:
    ___(6)___=0;
}
class SqlServerFactory:public IFactory{
public:
    IDepartment* CreateDepartment(){ return new SqlserverDepartment(); }
    //其余代码省略
};
class AccessFactory:public IFactory{
public:
    IDepartment* CreateDepartment(){ return new AccessDepartment(); }
    //其余代码省略
};
```

解析：

本题考查抽象工厂设计模式的概念及其应用。

抽象工厂设计模式的意图是：提供一个创建一系列相关或相互依赖的对象，而无须指出它们具体的类。在如下情况下应当考虑使用抽象工厂模式。

- 当一个系统要独立于它的产品的创建、组合和表示时。
- 当一个系统要由多个产品系列中的一个来配置时。
- 当需强调一系列相关的产品对象的设计以便进行联合使用时。
- 当想提供一组对象而不显示它们的实现过程，只显示它们的接口时。

抽象工厂设计模式的类图如图 6-3 所示，其中：

- Abstractory 为抽象工厂，声明抽象产品的方法。
- ConcreteFactory 为具体工厂，执行生成抽象产品的方法，生成一个具体的产品。
- Product A 和 Product B 为抽象产品，为一种产品声明接口。
- ProductA1、ProductA2、ProductB1、ProductB2 为具体产品，定义具体工厂生成的具体产品的对象，实现产品接口。

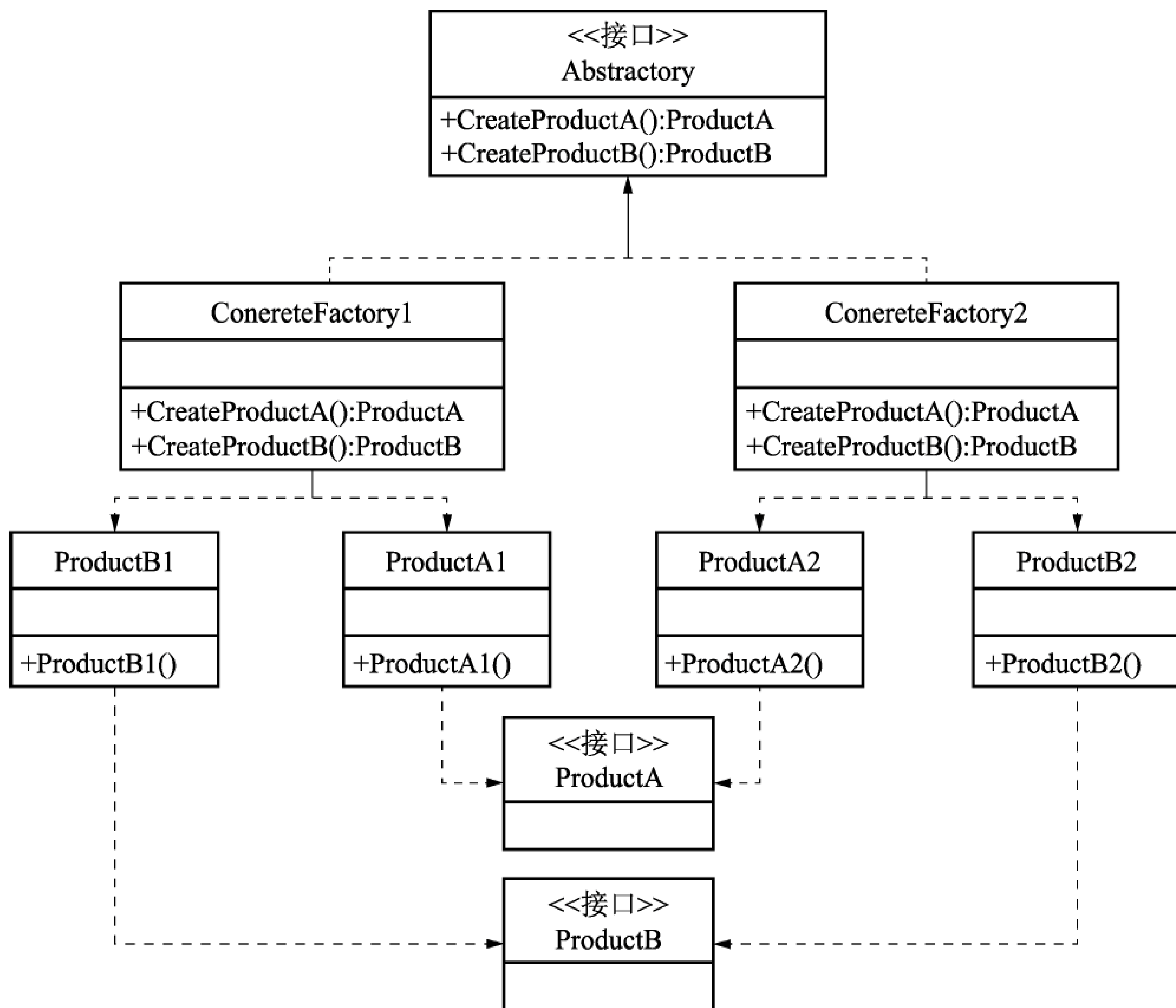


图 6-3　抽象工厂设计模式类图

图 6-2 中的 IFactory 对应图 6-3 中的 Abstractory，SqlserverFactory 和 AccessFactory 对应图 6-3 中的 ConcreteFactory，SqlserverDepartment 和 AccessDepartment 对应图 6-3 中的 ProductA1、ProductA2、ProductB1、ProductB2，而 IDepartment 对应图 6-3 中的 ProductA 和 ProductB。

由于类 IDepartment 的作用是为其子类提供接口，所以将其定义为抽象类。在 C++中，抽象类中至少包含一个纯虚函数的类，而纯虚函数是没有函数体的函数，其作用是为其子类提供统一的接口。若要使用纯虚函数，必须在子类中进行重置。空(1)处和空(2)处考查的是如何定义抽象类 IDepartment。从 IDepartment 的子类的方法中可以确定空(1)处应填入 virtual void Insert(Department* department)，空(2)处应填入 virtual Department GetDepartment(int id)。

空(3)处和空(4)处考查继承的概念和语法。由于 SqlserverDepartment 和 AccessDepartment

均继承 IDepartment，因此，空(3)处和空(4)处都应填入 public IDepartment。

由于所给程序中缺少有关抽象类 IFactory 的定义，因此空(5)处应定义类 IFactory，填入 class IFactory。抽象类 IFactory 中至少需包含一个纯虚函数的类，由其子类 SqlserverFactory 和 AccessFactory 中方法的定义可知，空(6)处应填入 virtual IDepartment* CreateDepartment()。

答案：

(1) virtual void Insert(Department* department)。

(2) virtual Department GetDepartment(int id)。

(3) public IDepartment。

(4) public IDepartment。

(5) class IFactory。

(6) virtual IDepartment* CreateDepartment()。

例 3　阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2012 年 5 月试题五)

【说明】(共 15 分)

某咖啡店卖咖啡时，可以根据顾客的要求在其中加入各种配料，咖啡店会根据所加入的配料来计算费用。咖啡店所供应的咖啡及配料的种类和价格如表 6-2 所示。

表 6-2　咖啡店所供应的咖啡及配料的种类和价格

咖　啡	价格/(元/杯)	配　料	价格/(元/杯)
蒸馏咖啡(Espresso)	25	摩卡(Mocha)	10
深度烘焙咖啡(DarkRoast)	20	奶泡(Whip)	8

现采用装饰器(Decorator)模式来实现计算费用的功能，得到如图 6-4 所示的类图。

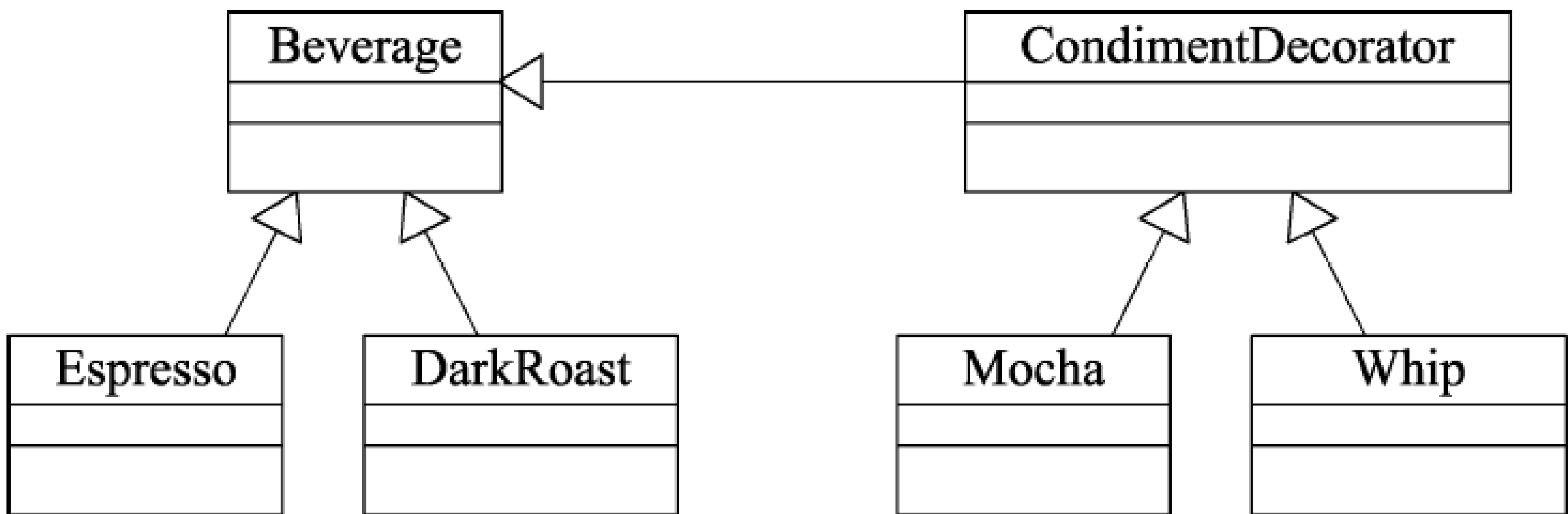


图 6-4　例 3 类图

【C++代码】

```
#include <iostream>
#include <string>
using namespace std;
const int ESPRESSO_PRICE = 25;
const int DRAKROAST_PRICE = 20;
const int MOCHA_PRICE = 10;
const int WHIP_PRICE = 8;
```

```
class Beverage { //饮料
  ___(1)___: string description;
public:
  ___(2)___ (){ return description; }
  ___(3)___;
};
class CondimentDecorator : public Beverage { //配料
protected:
  ___(4)___;
};
class Espresso : public Beverage { //蒸馏咖啡
public:
  Espresso () {description="Espresso"; }
  int cost(){return ESPRESSO_PRICE; }
};
class DarkRoast : public Beverage { //深度烘焙咖啡
public:
  DarkRoast(){ description = "DardRoast"; }
  int cost(){ return DRAKROAST_PRICE; }
};
class Mocha : public CondimentDecorator { //摩卡
public:
  Mocha(Beverage*beverage){ this->beverage=beverage; }
  string getDescription(){ return beverage->getDescription()+",Mocha"; }
  int cost(){ return MOCHA_PRICE+beverage->cost(); }
};
class Whip : public CondimentDecorator { //奶泡
public:
  Whip(Beverage*beverage) { this->beverage=beverage; }
  string getDescription() {return beverage->getDescription()+",Whip"; }
  int cost() { return WHIP_PRICE+beverage->cost(); }
};

int main() {
  Beverage* beverage = new DarkRoast();
  beverage=new Mocha(___(5)___);
  beverage=new Whip(___(6)___);
  cout<<beverage->getDescription()<<"￥"<<beverage->cost()<<endl;
  return 0;
}
```

编译运行上述程序，其输出结果为：

```
DarkRoast, Mocha, Whip ,￥38
```

解析：

由图 6-4 可知，Beverage 是基类，Espresso、DarkRoast、CondimentDecorator 是 Beverage 的派生类，Mocha、Whip 又是 CondimentDecorator 的派生类。

空(1)处应填入 description 的访问控制类型，可能为 private 或 protected。在 Beverage 的派生类 Espresso 的初始化函数中直接使用了 description，由此可知，在基类中，description 的访问控制类型为 protected。如果为 private，则在派生中不能使用。

在基类中先后动态建立了一个 DarkRoast 对象、Mocha 对象和 Whip 对象，调用初始化函数，输出在 Mocha 类和 Whip 类中分别调用了基类的 getDescription()和 cost()。

答案：

(1) protected。　(2) virtual string getDescription。　(3) virtual int cost()=0。

(4) Beverage* beverage。　(5) beverage。　(6) beverage。

例 4　阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2011 年 11 月试题五)

【说明】(共 15 分)

某大型商场内安装了多个简易的纸巾售卖机，自动售出 2 元钱一包的纸巾，且每次仅售出一包纸巾。纸巾售卖机的状态图如图 6-5 所示。

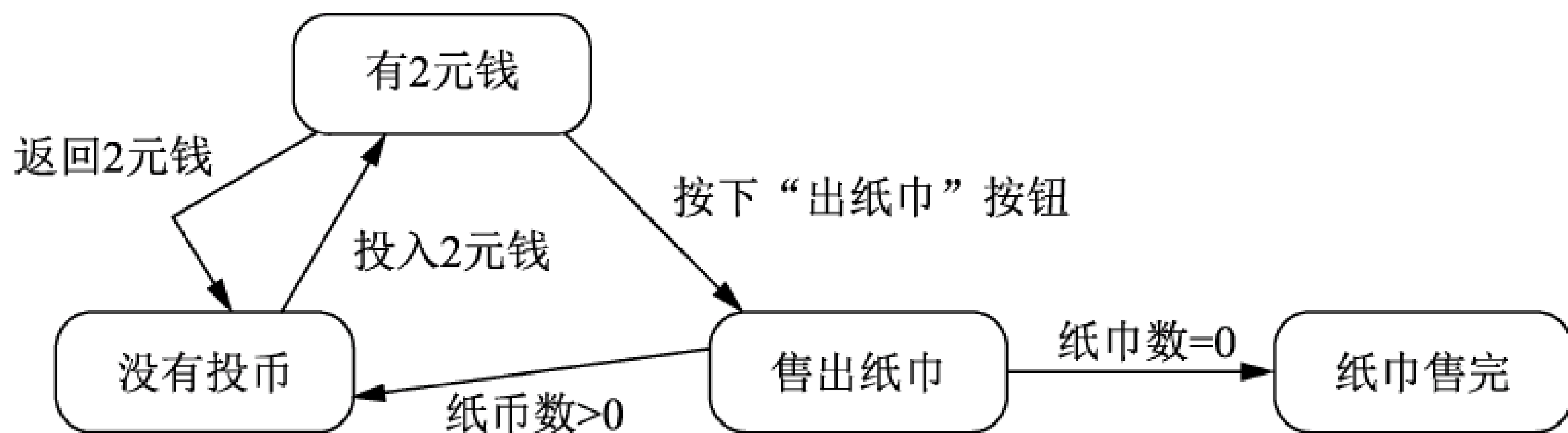


图 6-5　纸巾售卖机状态图

采用状态(State)模式来实现该纸巾售卖机，得到如图 6-6 所示的类图。其中类 State 为抽象类，定义了投币、退币、出纸巾等方法接口。类 SoldState、SoldOutState、NoQuarterState 和 HasQuarterState 分别对应图 6-5 中纸巾售卖机的 4 种状态：售出纸巾、纸巾售完、没有投币、有 2 元钱。

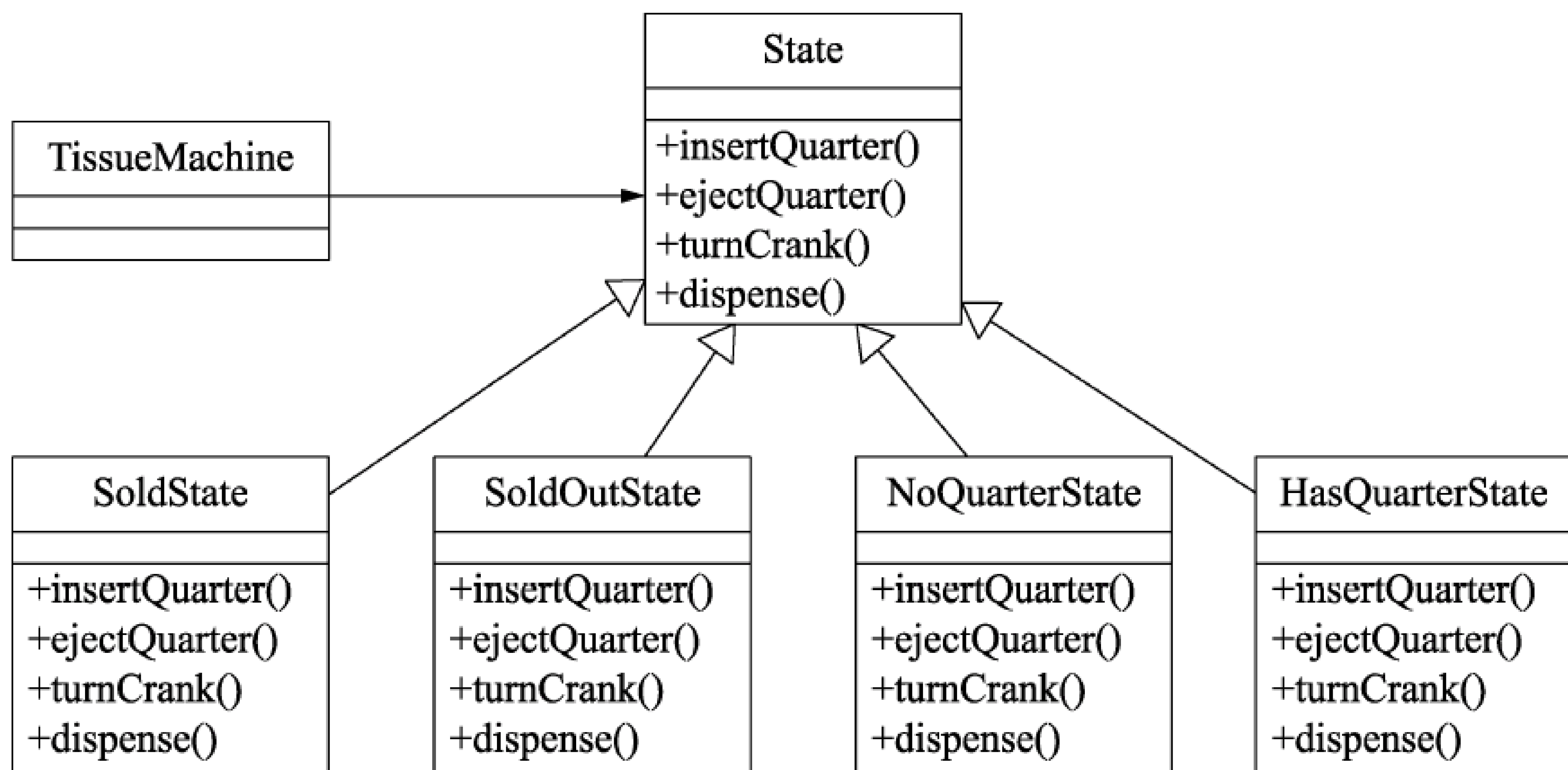


图 6-6　例 4 类图

【C++代码】

```
#include <iostream>
using namespace std;
//以下为类的定义部分
class TissueMachine;  //类的提前引用
class State{
public:
    virtual void insertQuarter()=0;    //投币
    virtual void ejectQuarter()=0;     //退币
    virtual void turnCrank()=0;        //按下“出纸巾”按钮
    virtual void dispense()=0;         //出纸巾
};
/*类 SoldOutState、NoQuarterState、HasQuarterState、SoldState 的定义省略，
每个类中均定义了私有数据成员 TissueMachine* tissueMachine*/
class TissueMachine{
private:
    (1) *soldOutState,*noQuarterState,*hasQuarterState,*soldState,*state;
    int count;  //纸巾数
public:
    TissueMachine(int number);
    void setState(State* state);
    State* getHasQuarterState();
    State* getNoQuarterState();
    State* getSoldState();
    State* getSoldOutState();
    int getCount();
    //其余代码省略
};

//以下为类的实现部分
void NoQuarterState::insertQuarter(){
    tissueMachine->setState( (2) );
}
void HasQuarterState::ejectQuarter(){
    tissueMachine->setState( (3) );
}
void SoldState::dispense(){
    if(tissueMachine->getCount()>0){
        tissueMachine->setState( (4) );
    }
    else{
        tissueMachine->setState( (5) );
    }
}
//其余代码省略
```

解析：

空(1)处：根据题意，本题使用的是状态模式，判断纸巾售卖机的状态，根据不同的状

态执行不同的动作。State 定义了纸巾售卖机所对应的一些状态，如售出纸巾、纸巾售完等。类 SoldOutState、NoQuarterState、HasQuarterState、SoldState 均由类 State 派生而来。

空(2)处：void insertQuarter()定义了一个“投币”的方法：在“没有投币”状态下，客户投币的方法。tissueMachine->setState 是改变纸巾售卖机的状态，此时，客户已投入 2 元钱，故将此时的状态改为“有 2 元钱”的状态，纸巾售卖机调用“有 2 元钱”状态的方法即可。

空(3)处：void ejectQuarter()定义了一个“退币”的方法：在“有 2 元钱”状态下，用户按下退币按钮，纸巾售卖机将此时的状态改为“没有投币”状态，故直接调用 getNoQuarterState()即可。

空(4)处：根据纸巾售卖机状态图可知，当售出纸巾并且纸巾数量仍大于 0 时，将返回“没有投币”状态，同空(3)处的分析，此处应调用 getNoQuarterState()。

空(5)处：根据纸巾售卖机状态图可知，当售出纸巾并且纸巾数量等于 0 时，将返回“纸巾售完”状态，此处用了 if…else…判断语句，当纸巾数量仍大于 0 时，返回“没有投币”状态，否则，纸巾数量一定等于 0。空(5)处是 else 下的一条语句，故此处是判断 count 为 0 时纸巾售卖机的状态，显然，应调用 getSoldOutState()。

答案：

(1) State。

(2) tissueMachine->getHasQuarterState()或 new HasQuarterState。

(3) tissueMachine->getNoQuarterState()或 new NoQuarterState。

(4) tissueMachine->getNoQuarterState()或 new NoQuarterState。

(5) tissueMachine->getSoldOutState()或 new SoldOutState。

例 5　阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2011 年 5 月试题五)

【说明】(共 15 分)

某饭店在不同的时段提供多种不同的餐饮，其菜单结构图如图 6-7 所示。

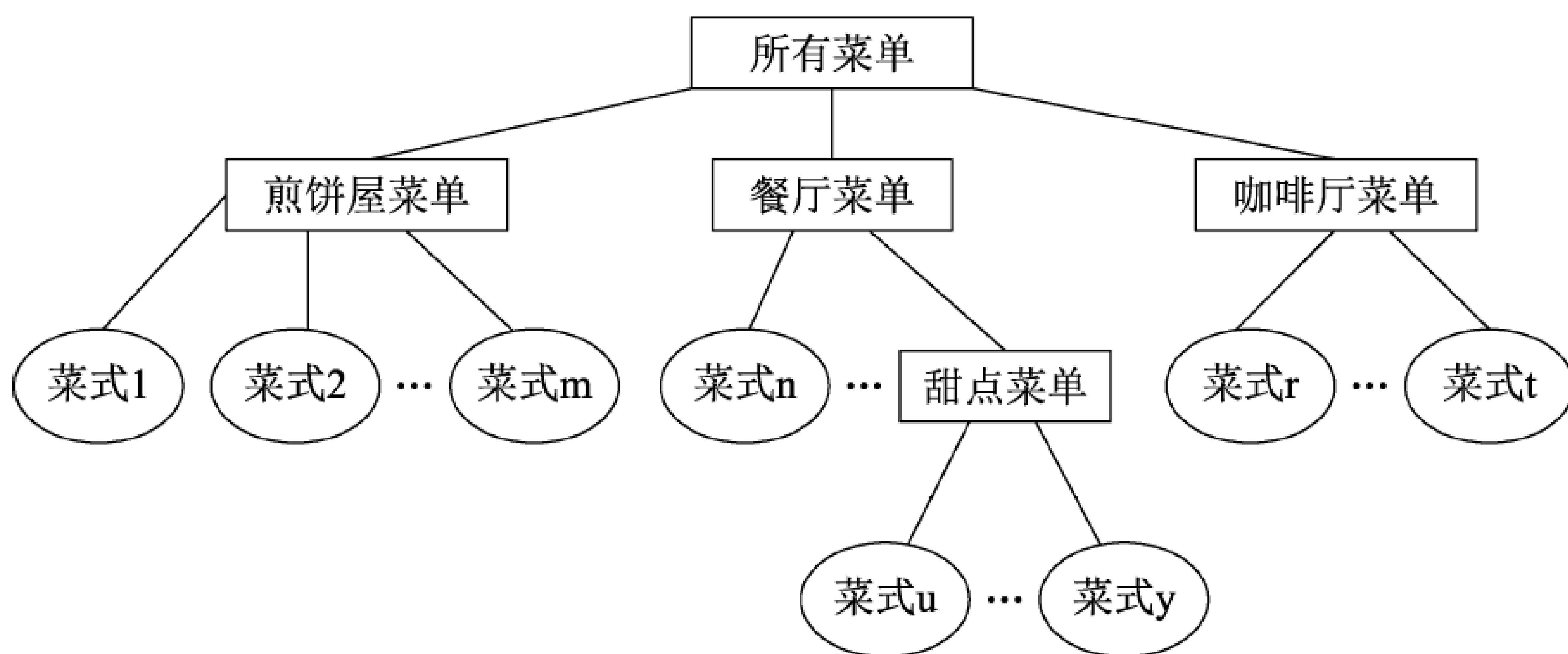


图 6-7　菜单结构图

现在采用组合(Composition)模式来构造该饭店的菜单，使得饭店可以方便地在其中添加新的餐饮形式，得到如图 6-8 所示的类图。其中 MenuComponent 为抽象类，定义了添加(add)新菜单和打印饭店所有菜单信息(print)的方法接口。类 Menu 表示饭店提供的每种餐饮形式

的菜单，如煎饼屋菜单、咖啡屋菜单等。每种菜单中都可以添加子菜单，例如图 6-7 中的甜点菜单。类 MenuItem 表示菜单中的菜式。

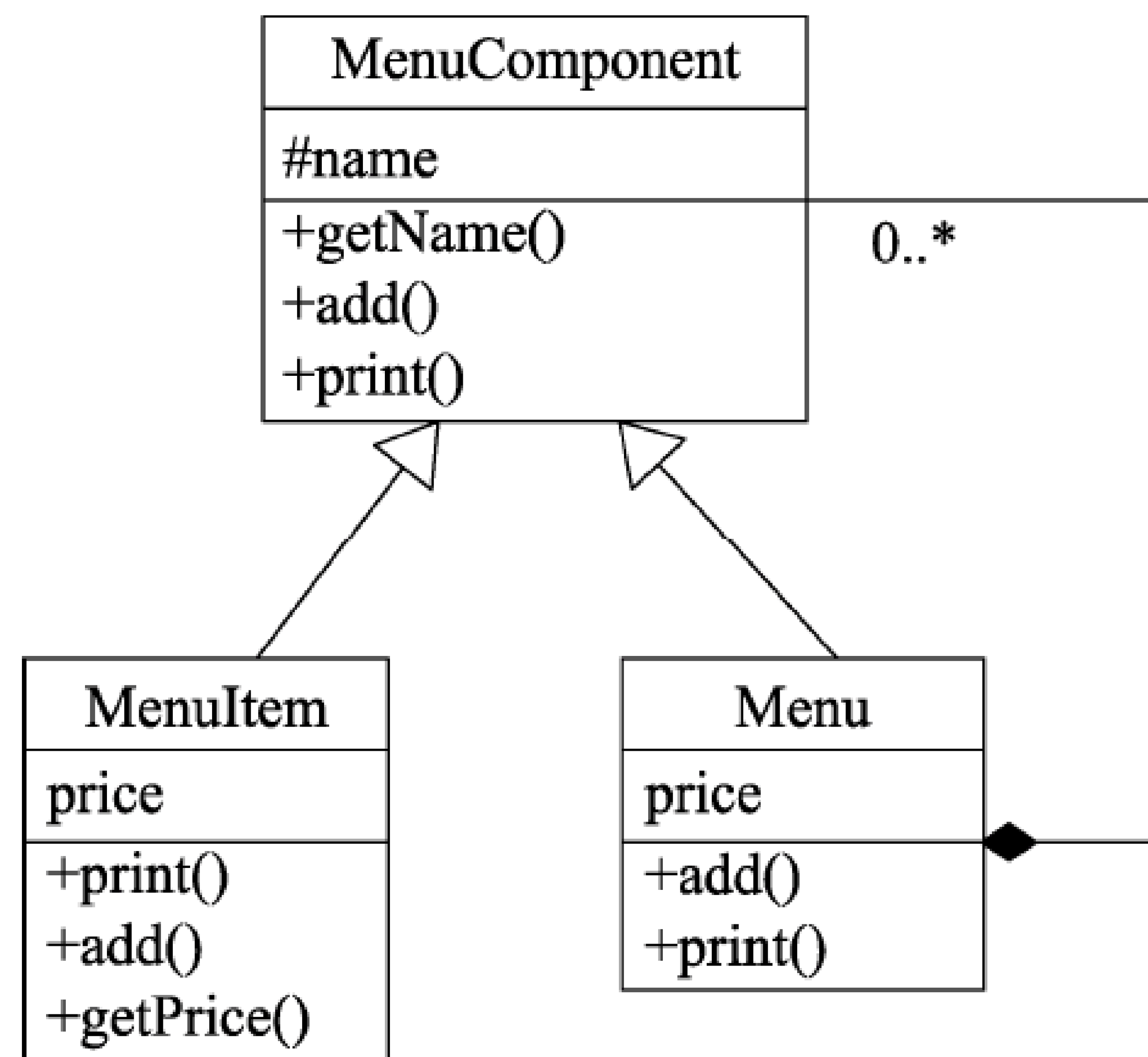


图 6-8 例 5 类图

【C++代码】

```
#include <iostream>
#include <list>
#include <string>
using namespace std;
class MenuComponent{
protected: string name;
public:
    MenuComponent(string name){ this->name=name; }
    string getName(){ return name; }
    __(1)__; //添加新菜单
    virtual void print()=0;  //打印菜单信息
};
class MenuItem:public MenuComponent{
private:double price;
public:
    MenuItem(string name, double price):MenuComponent(name)
{ this->price=price;}
    double getPrice(){return price;}
    void add(MenuComponent* menuComponent){return;}  //添加新菜单
    void print(){cout<<"  "<<getName()<<","<<getPrice<<endl;}
};
class Menu:public MenuComponent{
private: list<__(2)__>menuComponents;
public:
    Menu(string name):MenuComponent(name){}
    void add(MenuComponent* menuComponent)  //添加新菜单
    {__(3)__; }
    void print(){
        cout<<"\n"<<getName()<<"\n-----------------"<<endl;
```

```
        std::list<MenuComponent *>::iterator iter;
        for(iter=menuComponents.begin(); iter!=menuComponents.end(); iter++)
            __(4)__->print();
    }
};
void main(){
    MenuComponent* allMenus=new Menu("ALL MENUS");
    MenuComponent* dinerMenu=new Menu("DINER MENU");
    … //创建更多的 Menu 对象，此处代码省略
    allMenus->add(dinerMenu); //将 dinerMenu 添加到餐厅菜单中
    … //为餐厅增加更多的菜单，此处代码省略
    __(5)__->print(); //打印饭店所有菜单信息
}
```

解析：

MenuComponent 是虚基类，其中要定义添加新菜单纯虚函数，函数体在其派生类 MenuItem 和 Menu 中实现。空(1)处应填入 virtual void add(MenuComponent* menuComponet)=0。

在类 Menu 中定义了列表 menuComponents，用于保存添加的新菜单，由 std::list<MenuComponent *>::iterator iter 可知列表的类型为 MenuComponent *，故空(2)处应填入 MenuComponent*。

添加菜单时，可以从 menuComponents 列表的前面添加，也可以从 menuComponents 列表的后面添加，但通过菜单的输出函数可知，菜单是从列表的后面添加的，因此空(3)处应填入 menuComponents->push_back(menuComponent)。

iter 是迭代器，用于遍历菜单列表 menuComponents，空(4)处应填入(*iter)。

在主函数中，打印饭店所有菜单信息的函数调用为 allMenus->print()。

答案：

(1) virtual void add(MenuComponent* menuComponet)=0。

(2) MenuComponent*。

(3) menuComponents->push_back(menuComponent)。

(4) (*iter)。

(5) allMenus。

例 6　阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2010 年 11 月试题五)

【说明】(共 15 分)

某公司的组织结构图如图 6-9 所示，现采用组合(Composition)设计模式来构造该公司的组织结构，得到如图 6-10 所示的类图。

其中 Company 为抽象类，定义了在组织结构图上添加(Add)和删除(Delete)分公司/办事处或者部门的方法接口。类 ConcreteCompany 表示具体的分公司或者办事处，分公司或办事处下可以设置不同的部门。类 HRDepartment 和 FinanceDepartment 分别表示人力资源部和财务部。

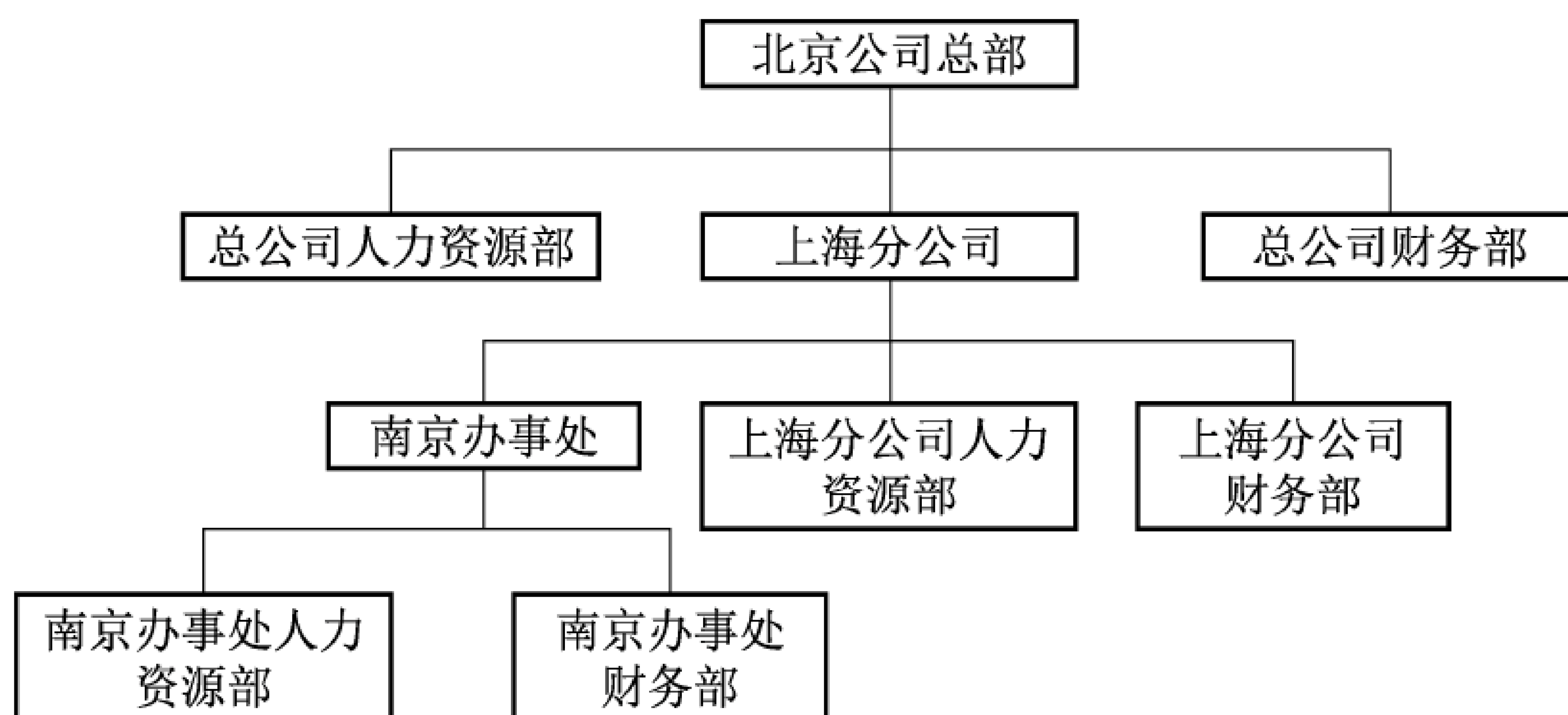


图 6-9　组织结构图

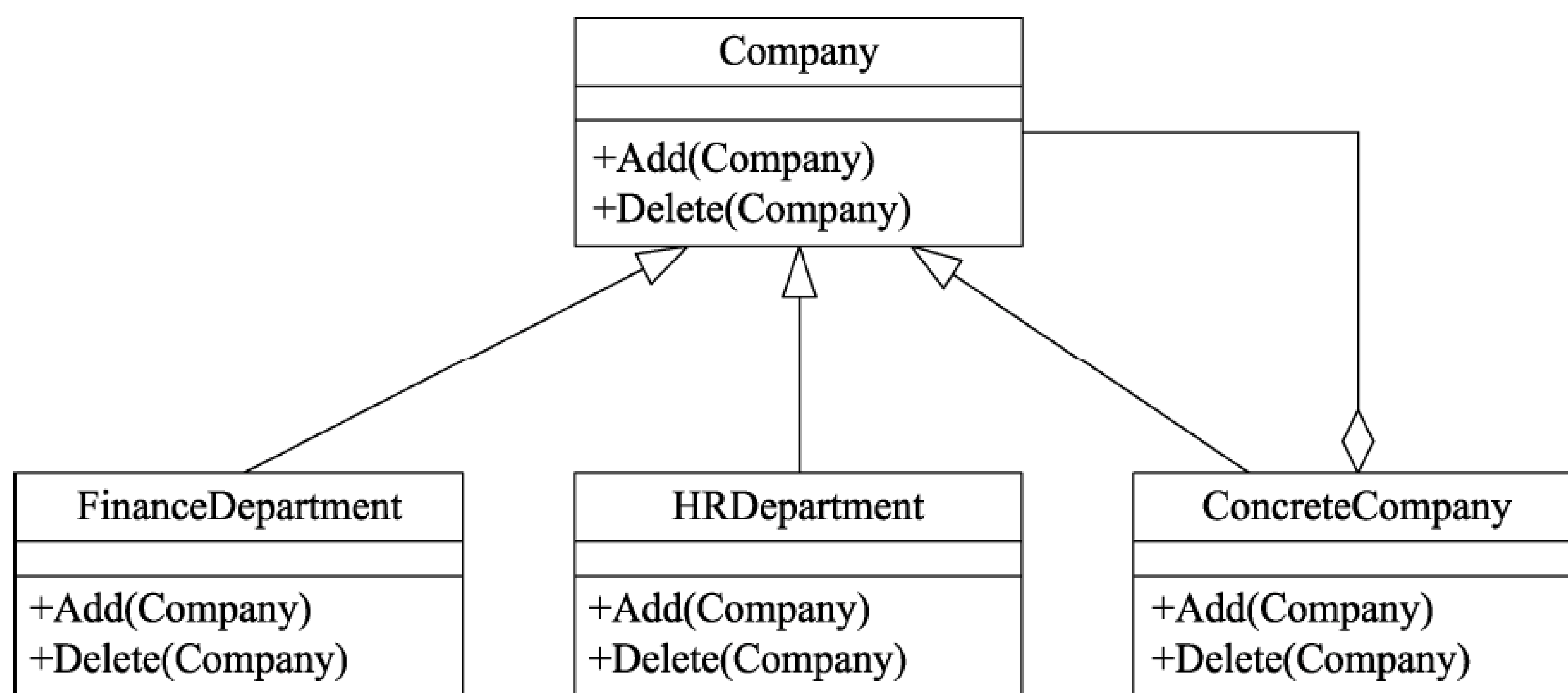


图 6-10　例 6 类图

【C++代码】

```
#include <iostream>
#include <list>
#include <string>
using namespace std;
class Company{//抽象类
protected:
    string name;
public:
    Company(string name){___(1)___=name;}
    ___(2)___;//增加子公司、办事处或部门
    ___(3)___;//删除子公司、办事处或部门
};
class ConcreteCompany: public Company{
private:
    list<___(4)___>children;//存储子公司、办事处或部门
public:
    ConcreteCompany(string name):Company(name){}
    void Add(Company* c){___(5)___.push back(c);}
    void Delete(Company* c){___(6)___.remove(c);}
};
```

```
class HRDepartment: public Company{
public:
    HRDepartment(string name):Company(name){}      //其他代码省略
};
class FinanceDepartment: public Company{
public:
    FinanceDepartment(string name):Company(name){} //其他代码省略
};
void main(){
    ConcreteCompany *root=new ComcreteCompany("北京总公司");
    root->Add(new HRDepartment("总公司人力资源部"));
    root->Add(new FinanceDepartment("总公司财务部"));
    ConcreteCompany *comp=new ConcreteCompany("上海分公司");
    comp->Add(new HRDepartment("上海分公司人力资源部"));
    comp->Add(new FinanceDepartment("上海分公司财务部"));
     (7) ;
    ConcreteCompany *comp1=new ConcreteCompany("南京办事处");
    comp1->Add(new HRDepartment("南京办事处人力资源部"));
    comp1->Add(new FinanceDepartment("南京办事处财务部"));
     (8) ;//其他代码省略
}
```

解析：

初始化函数中，将形参的值赋给成员 name，形参的变量名和成员变量的名称相同，需要使用 this 指针指示被赋值的 name 是类的成员。

增加(删除)子公司、办事处或部门用到的函数是 Add 和 Delete。由于 Company 是抽象类，并作为 ConcreteCompany、HRDepartment、FinanceDepartment 的基类，ConcreteCompany、HRDepartment 继承了其父类的 Add 和 Delete 操作，因此在基类中要将 Add 和 Delete 设置为纯虚函数。

类 ConcreteCompany 表示具体的分公司或者办事处，其中的成员 children 用来存储子公司、办事处或部门，其数据类型应为 Company*，当进行增加操作时，要把增加的对象存储在 children 的最后；当进行删除操作时，则需要从 children 中将对应的对象移除。

空(7)处的操作是把上海分公司这个对象加入到北京公司总部中。

空(8)处的操作是将南京办事处这个对象加入到上海分公司中。

答案：

(1) this->name。

(2) virtual void Add(Company* c)=0。

(3) virtual void Delete(Company* c)=0。

(4) Company*。

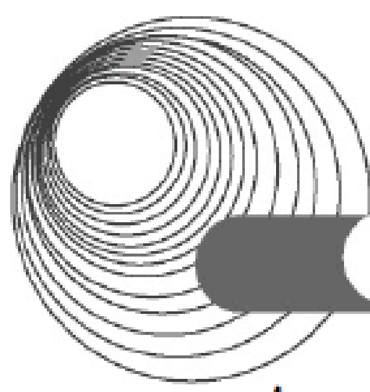
(5) children。

(6) children。

(7) root->Add(comp)。

(8) comp->Add(comp1)。

例 7　阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2010



年 5 月试题五)

【说明】(共 15 分)

某软件公司现欲开发一款飞机飞行模拟系统，该系统主要模拟不同种类飞机的飞行特征与起飞特征。需要模拟的飞机种类及其特征如表 6-3 所示。

表 6-3　需要模拟的飞机种类及其特征

飞机种类	起飞特征	飞行特征
直升机(Helicopter)	垂直起飞(VerticalTakeOff)	亚音速飞行(SubSonicFly)
客机(AirPlane)	长距离起飞(LongDistanceTakeOff)	亚音速飞行(SubSonicFly)
歼击机(Fighter)	长距离起飞(LongDistanceTakeOff)	超音速飞行(SuperSonicFly)
鹞式战斗机(Harrier)	垂直起飞(VerticalTakeOff)	超音速飞行(SuperSonicFly)

为支持将来模拟更多种类的飞机，采用策略设计模式(Strategy)设计的类图如图 6-11 所示。

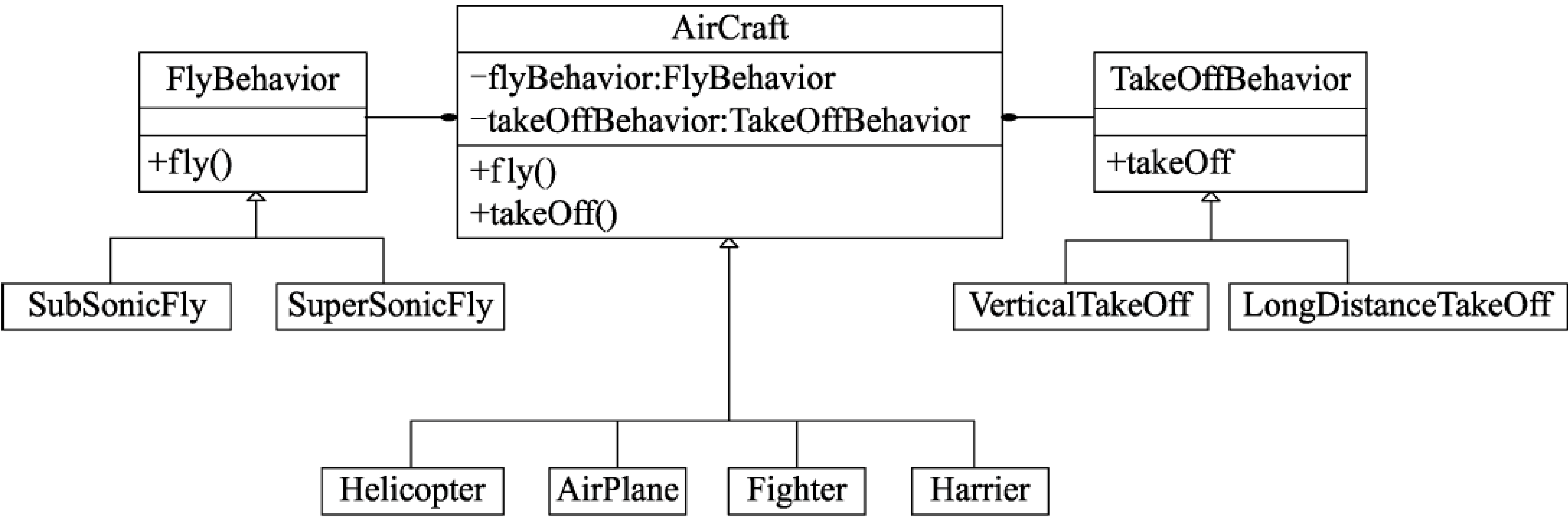


图 6-11　例 7 类图

图 6-11 中，AirCraft 为抽象类，描述了抽象的飞机，而类 Helicopter、AirPlane、Fighter 和 Harrier 分别描述具体的飞机种类，方法 fly()和 takeOff()分别表示不同飞机都具有飞行特征和起飞特征；类 FlyBehavior 与 TakeOffBehavior 为抽象类，分别用于表示抽象的飞行行为与起飞行为；类 SubSonicFly 与 SuperSonicFly 分别描述亚音速飞行和超音速飞行的行为；类 VerticalTakeOff 与 LongDistanceTakeOff 分别描述垂直起飞与长距离起飞的行为。

【C++代码】

```
#include <iostream>
using namespace std;
class FlyBehavior {
 public: virtual void fly() = 0;
};
class SubSonicFly:public FlyBehavior{
 public: void fly(){ cout << "亚音速飞行！" << endl; }
};
class SuperSonicFly:public FlyBehavior{
 public: void fly(){ cout << "超音速飞行！" << endl; }
};
```

```
class TakeOffBehavior {
 public: virtual void takeOff() = 0;
};
class VerticalTakeOff:public TakeOffBehavior{
 public: void takeOff(){ cout << "垂直起飞! " << endl; }
};
class LongDistanceTakeOff:public TakeOffBehavior {
 public: void takeOff (){ cout << "长距离起飞! " << endl; }
};
class AirCraft{
protected:
___(1)___;
___(2)__;
public:
 void fly(){___(3)___; }
 void takeOff() {___(4)___; };
};
class Helicopter: public AirCraft {
public:
 Helicopter (){
 flyBehavior = new___(5)___;
 takeOffBehavior = new___(6)___;
}
___(7)___{
 if(!flyBehavior) delete flyBehavior;
 if(!takeOffBehavior) delete takeOffBehavior;
 }
};
//其他代码省略
```

解析：

本题考查设计模式中的策略设计模式。

从本题的叙述中可以看出，存在 4 种不同的飞机类型，但每种飞机类型的起飞特征和飞行特征并不完全相同，这就使得我们很难采用比较直接的方法来实现重用。例如，定义一个抽象的飞机类，实现飞机的起飞特性，然后 4 种飞机直接重用该特征。但是，我们可以观察到，尽管飞机的起飞特征和飞行特征有所不同，有一点可以肯定的是，每一种飞机都具备了飞行特征和起飞特征。因此，可以抽象出一个飞机类，其中含有飞行特征与起飞特征，但关于两个特征的实现要单独抽取出来，所以又形成了 FlyBehavior 类和 TakeOffBehavior 类，分别表示抽象的飞行和起飞特征，而这两个类的子类则分别实现不同的起飞和飞行特征，最终转化为，在创建一个具体的飞机时，给其配上不同的起飞特征和飞行特征即可。

本题中的空(1)处和空(2)处应该填写成员变量，根据图 6-11 可知，此处应该表示的是飞行和起飞特征变量，在 C++中可以采用指针来表示。空(3)处和空(4)处需要实现飞行与起飞特征，但 AirCraft 是抽象的类，所以把实现代理给指针变量。Helicopter 类需要指定由父类继承而来的成员变量的初始值，因为 Helicopter 的特征是垂直起飞和亚音速飞行，因此生成这两个特征的对象，分别赋值给 flyBehavior 和 takeOffBehavior 变量。

答案：

(1) FlyBehavior*flyBehavior。

(2) TakeOffBehavior*takeOffBehavior。

(3) flyBehavior->fly()。

(4) takeOffBehavior->takeoff()。

(5) SubSonicFly()。

(6) VerticalTakeOff()。

(7) ~Helicopter()。

注：空(1)处与空(2)处答案可互换。

例 8 阅读以下说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2013 年 11 月试题五)

【说明】

欲开发一个绘图软件，要求使用不同的绘图程序绘制不同的图形。以绘制直线和圆形为例，对应的绘图程序如表 6-4 所示。

表 6-4　不同的绘图程序

	DP1	DP2
绘制直线	draw_a_line(x1,y1,x2,y2)	drawline(x1,x2,yl,y2)
绘制圆	draw_a_circle(x,y,r)	drawcircle(x,y,r)

该绘图软件的扩展性要求，将不断扩充新的图形和新的绘图程序。为了避免出现类爆炸的情况，现采用桥接(Bridge)模式来实现上述要求，得到如图 6-12 所示的类图。

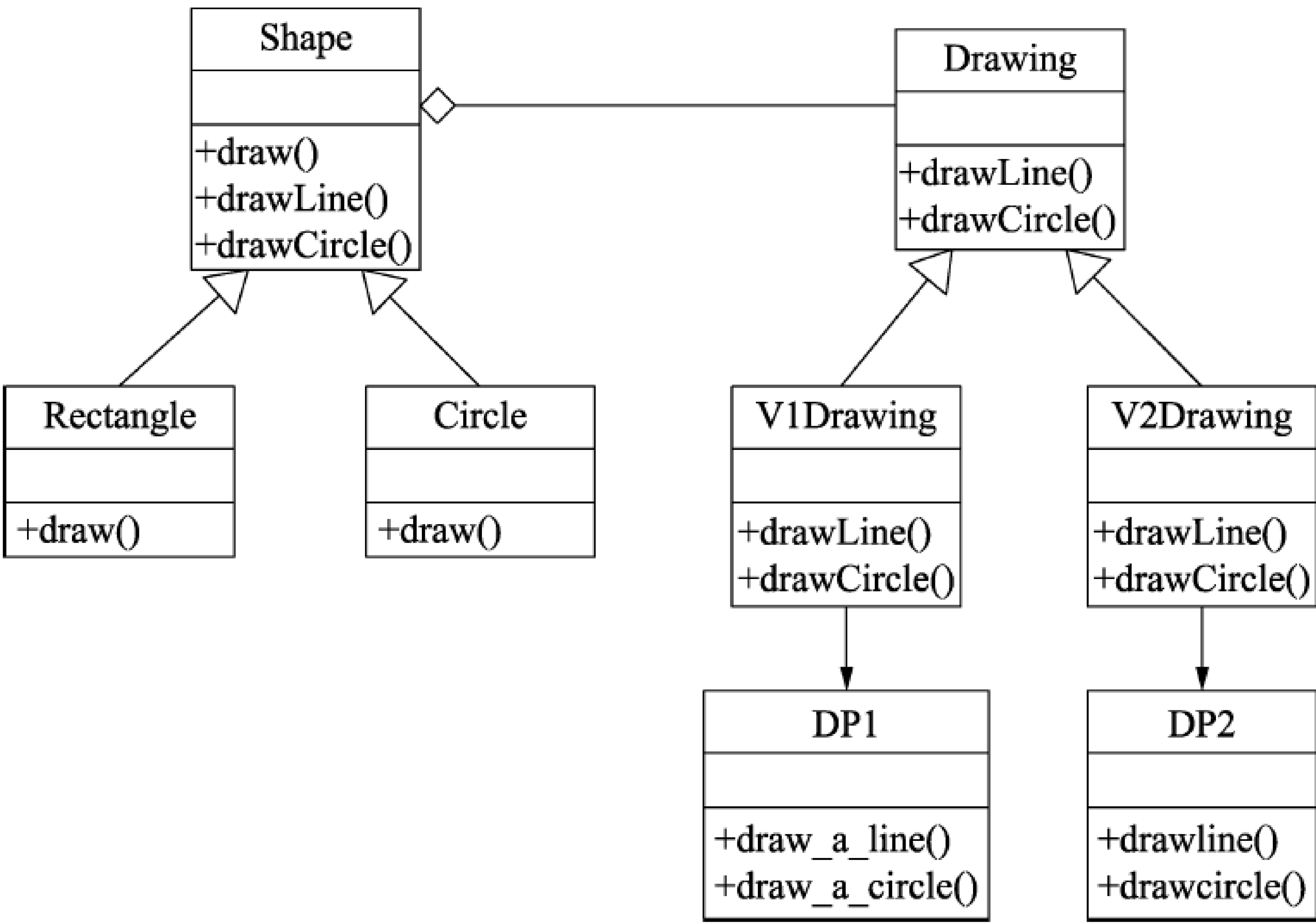


图 6-12　某绘图软件类图

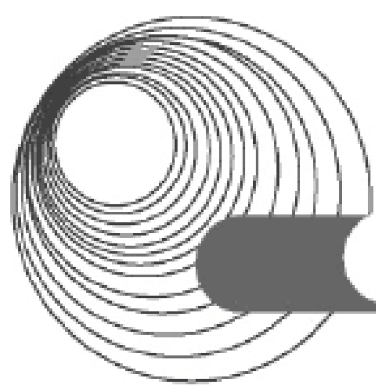
【C++代码】

```
class DP1{
public:
```

```
    static void draw_a_line(double x1,double y1,double x2,double y2) {/* 代码省略 */}
    static void draw_a_circle(double x,double y,double r) {/* 代码省略 */}
};
class DP2{
public:
    static void drawline(double x1,double x2,double y1,double y2) {/* 代码省略 */}
    static void drawcircle(double x,double y,double r) {/* 代码省略 */}
};
class Drawing{
public:
____(1)____;
____(2)____;
};
class V1Drawing:public Drawing{
public:
    void drawLine(double x1,double y1,double x2,double y2) {/* 代码省略 */}
    void drawCircle(double x,double y,double r) {____(3)____;}
};
class V2Drawing:public Drawing{
public:
    void drawLine(double x1,double y1,double x2,double y2) {/* 代码省略 */}
    void drawCircle(double x,double y,double r) {____(4)____;}
};
class Shape{
public:
  ____(5)____;
    Shape(Drawing *dp){_dp=dp;}
    void drawLine(double x1,double y1,double x2,double y2){
        _dp->drawLine(x1,y1,x2,y2);}
void drawCircle(double x,double y,double r){_dp->drawCircle(x,y,r);}
private:    Drawing *_dp;
};

class Rectangle:public Shape{
public:
    void draw(){    /* 代码省略 */    }
    //其余代码省略
};

class Circle:public Shape{
private: double _x,_y,_r;
public:
    Circle(Drawing *dp,double x,double y,double r):___(6)___{
    _x=x;_y=y;_r=r;}
    void draw(){    drawCircle(_x,_y,_r);    }
};
```



解析：

本题考查 Bridge 桥接模式的概念及应用，将抽象与其实现解耦，使它们都可以独立地变化。大致意思是说：将一组实现与另一组使用它们的对象分离。这里的实现指的是抽象类及其派生类用来实现自己的对象(而不是抽象类的派生类，这些派生类被称为具体类)。

Drawing 是一个虚拟基类，里面包含了希望不同策略实现的算法，派生类 V1Drawing、V2Drawing 都派生自 Drawing，对基类中的希望实现的算法都作了具体实现，且它们都含有 drawLine 和 drawCircle 函数，所以 Draw 类中缺失的应该是这两个算法，于是 (1) 处填 virtual void drawLine(double x1,double y1,double x2,double y2=0， (2) 处填 virtual void drawCircle (double x,double y,double r)=0。DP1 和 DP2 中包含了绘制 Line 和 Circle 的具体实现的 Static 方法，因此在 V1Drawing、V2Drawing 类中可以直接进行调用它们。 (3) 处填 DP1::draw_a_circle(x,y,r)， (4) 处填 DP2::drawCircle(x,y,r)。Shape 类派生出 Rectangle 和 Circle 类，里面都含有 draw 方法，但是具体 draw 方法的实现却不相同，所以 (5) 处填 virtual void draw()=0。最后 Circle 的构造函数初始化了后四个变量，还有继承自 shape 的变量_dp 未做初始化，因此调用 shape 的构造函数做初始化， (6) 处填 shape(dp)。

答案：

(1) virtual void drawLine(double x1,double y1,double x2,double y2)=0。

(2) virtual void drawCircle(double x,double y,double r)=0。

(3) DP1::draw_a_circle(x,y,r)。

(4) DP2::drawCircle(x,y,r)。

(5) virtual void draw()=0。

(6) Shape(dp)。

例 9 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2014 年 11 月试题五)

【说明】

某灯具厂商欲生产一个灯具遥控器，该遥控器具有 7 个可编程的插槽，每个插槽都有开关按钮，对应着一个不同的灯。利用该遥控器能够统一控制房间中该厂商所有品牌灯具的开关，现采用 Command(命令)模式实现该遥控器的软件部分。Command 模式的类图如图 6-13 所示。

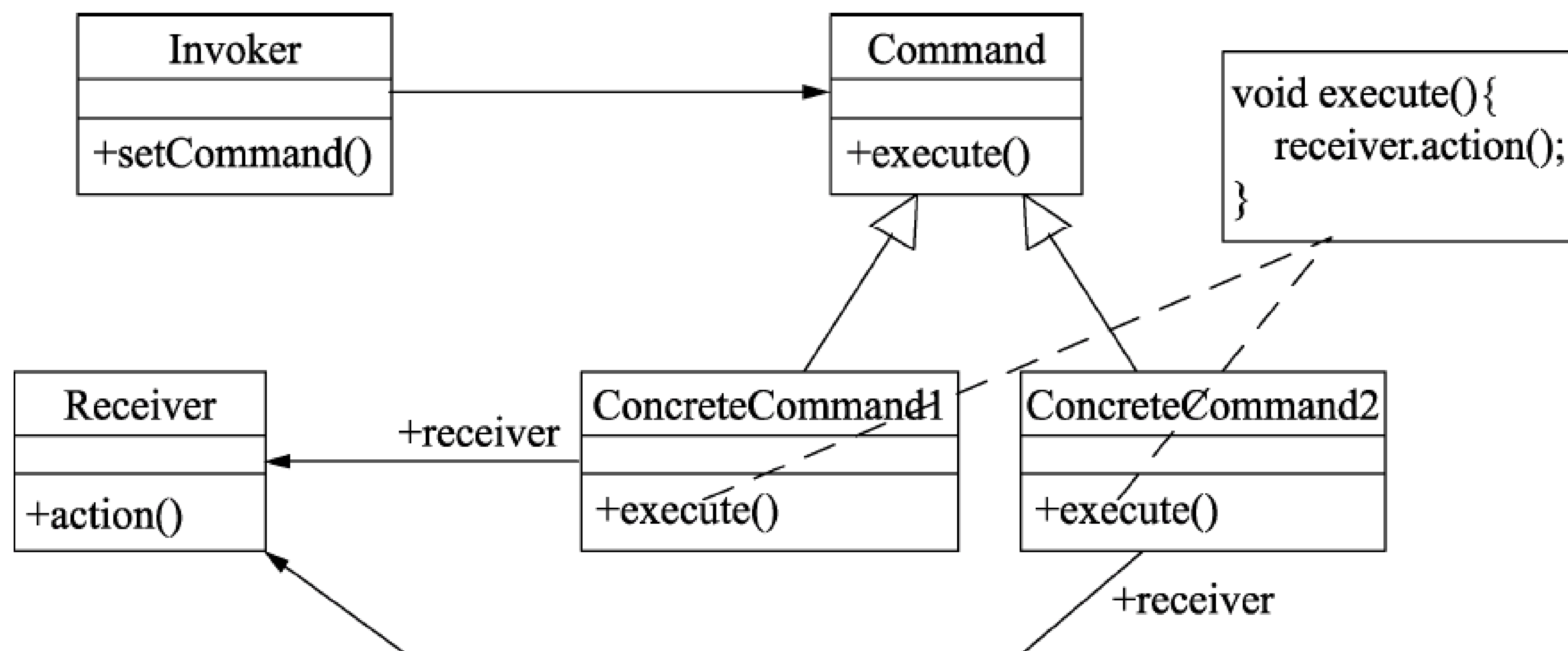


图 6-13　Command 模式类图

【C++代码】

```
class Light{
public:
  Light(string name){/*代码省略*/}
void on(){/*代码省略*/}  //开灯
void off(){/*代码省略*/}  //关灯
};
class Command{
public:
 (1) ;
};
class LightOnCommand:public Command{//开灯命令
private:
  Light* light;
public:
  LightOnCommand(Light*light){this->light=light;}
  void execute(){ (2) ;}
};
class LightOffCommand:public Command{//关灯命令
private:
  Light* light;
public:
  LightOffCommand(Light*light){this->light=light;}
  void execute(){ (3) ;}
};
class RemoteControl{ //遥控器
private:
Command* onCommands[7];
Command* offCommands[7];
public:
RemoteControl() { /*代码省略*/ }
void setCommand(int slot Command* onCommand, Command* offCommand) {
(4) =onCommand;
(5) =offCommand;
}
void onButtonWasPushed(int slot) { (6) ;}
void offButtonWasPushed(int slot) { (7) ;}
};
int main() {
RemoteControl * remoteControl=new RemoteControl();
Light* livingRoomLight=new Light("Living Room");
Light* kitchenLight=new Light("kitchen");
LightOnCommand* livingRoomLightOn= new LightOnCommand(livingRoomLight);
LightOffCommand* livingRoomLightOff=new LightOffCommand(livingRoomLight);
LightOnCommand* kitchenLightOn=new LightOnCommand(kitchenLight);
LightOffCommand* kitchenLightOff=new LightOffCommand(kitchenLight);
remoteControl->setCommand(0, livingRoomLightOn, livingRoomLightOff);
remoteControl->setCommand(1, kitchenLightOn, kitchenLightOff);
```

```
remoteControl->onButtonWasPushed(0);
remoteControl->offButtonWasPushed(0);
remoteControl->onButtonWasPushed(1);
remoteControl->offButtonWasPushed(1);
/*其余代码省略*/
return 0;
}
```

解析：

本题考查 Command(命令)模式的实现，难度较小，根据类图和已有代码可写出空缺的代码，书写方式注意 Java 和 C++的区别。

由图 6-13 可知，Command 类中有个 execute 函数，下面两个类继承了该类的 execute 函数，故该函数为虚函数实现多态，前面应有 virtual，第(1)空答案得到。第(2)空和第(3)空为 light 灯分别开关的命令，在 on 类和 off 类中，故为 light->on()和 light->off()。在 setCommand 里面，传了 slot 进来，表示 slot 个灯泡。第(4)空表示第 slot 个灯泡开，第(5)空表示第 slot 个灯泡关。第(6)空和第(7)空执行开关灯函数，为 onCommands[slot]->execute()和 offCommands[slot] -> execute()。

第(4)空和第(5)空都为 setCommand 函数里的内容，slot 为 int 型，控制第几个灯泡亮和关，由 Command* onCommand 和 Command* offCommand，又有 Command* onCommands[7]和 Command* offCommands[7]，故 Command 函数应为 onCommands，第(4)空和第(5)空分别为 onCommands[slot]、offCommands[slot]。

同理，第(6)空第(7)空为 onCommands[slot]-> execute()和 offCommands[slot]-> execute()。

答案：

(1) virtual void execute()=0。

(2) light->on()。

(3) light->off()。

(4) onCommands[slot]。

(5) offCommands[slot]。

(6) onCommands[slot]->execute()。

(7) offCommands[slot]->execute()。

例 10 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2014 年 5 月试题五)

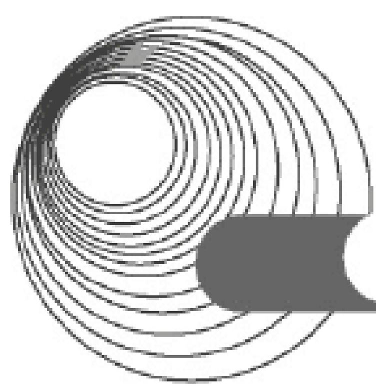
【说明】

某实验室欲建立一个实验室环境监测系统，能够显示实验室的温度、湿度以及洁净度等环境数据。当获取到最新的环境测量数据时，显示的环境数据能够更新。现在采用观察者(Observer)模式来开发该系统，观察者模式的类图如图 6-14 所示。

图 6-14　观察者模式的类图

【C++代码】

```
#include <iostream>
#include <vector>
using namespace std;
class Observer {
public:
virtual void update(float temp, float humidity, float cleanness)=0;
};
class Subject {
public:
virtual void registerObserver(Observer* o)=0;//注册对主题感兴趣的观察者
virtual void removeObserver(Observer* o)=0;//删除观察者
virtual void notifyObservers()= 0;//当主题发生变化时通知观察者
}
class EnvironmentData : public____(1)____{
private:
vector<Observer*> observers;
float temperature, humidity, cleanness;
public:
void registerObserver(Observer* o) { observers.push_back(o); }
void removeObserver(Observer* o){/*代码省略*/}
void notifyObservers( ) {
for(vector<Observe*>::const_iterator it= observers.begin();    it !=
observers.end(); it++)
{___(2)___; }
}
void measurementsChanged(){ ___(3)___;}
void setMeasurements(float temperature, float humidity, float cleanness) {
this->temperature=temperature;
this->humidity=humidity;
this->cleanness=cleanness;
___(4)___;
}
};
```




```
class CurrentConditionsDisplay : public__(5)__{
private:
float temperature, humidity, cleanness;
Subject* envData;
public:
CurrentConditionsDisplay(Subject* envData) {
this->envData=envData;
__(6)__;
}
void update(float temperature, float humidity, float cleanness)
{this->temperature=temperature;
This->humidity = humidity;
This->cleanness = cleanness;
display( );
}
void display( ){/*代码省略*/}
};
int main( ){
EnvironmentData* envData=new EnvironmentData();
CurrentConditionsDisplay* currentDisplay=new CurrentConditionsDisplay(envData);
envData->setMeasurements(80, 65, 30.4f);
return 0;
}
```

解析：

EnvironmentData 是环境数据，也就是我们要监测的对象，即主题(Subject)，因此(1)处为 Subject。

(2)处为通知观察者，因此遍历观察者容器，遍历到一个观察者对象，则更新该观察者的数据，即调用观察者的 update()方法。

当环境数据变化时，需要通知观察者，因此(4)处是调用环境变化方法 measurementsChanged()，通过此方法通知观察者更新数据，因此(3)处为 notifyObservers()。

根据 CurrentConditionsDisplay 类中的 update()方法可知：CurrentConditionsDisplay 是个观察者，因此(5)处为 Observer。

(6)是将观察者添加到主题中去。

答案：

(1) Subject。

(2) (*it)->update(temperature,humidity,cleanness)。

(3) notifyObservers()。

(4) measurementsChanged()。

(5) Observer()。

(6) this->envData-> registerObserver(this)。

例 11 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2015 年 5 月试题五)

【说明】

某图书管理系统中管理着两种类型的文献：图书和论文。现在要求统计所有馆藏文献的总页码(假设图书馆中有一本 540 页的图书和两篇各 25 页的论文，那么馆藏文献的总页码就是 590 页)。采用 Visitor(访问者)模式实现该要求，得到如图 6-15 所示的类图。

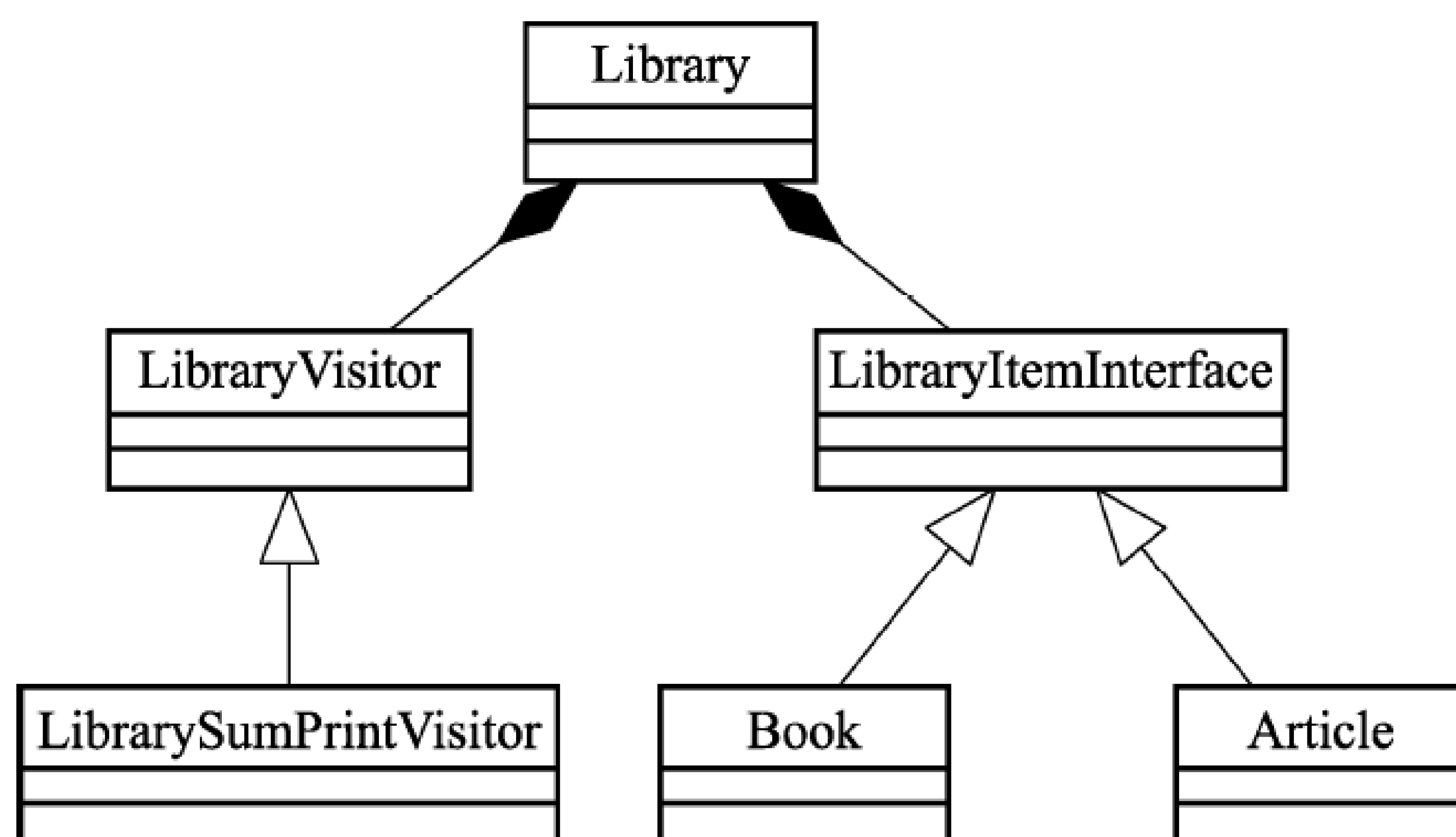


图 6-15　Visitor 模式类图

【C++代码】

```
class LibraryVisitor;
class LibraryItemInterface{
public:
____(1)____;
};
class Article : public LibraryItemInterface {
    private:
    string  m_title;        //论文名
    string  m_author;       //论文作者
    int m_start_page;
    int m_end_page;
public:
    Article(string p_author, string p_title, int p_start_page,int p_end_page);
    int getNumberOfPages();
    void accept(Library Visitor* visitor);
};
class Book : public LibraryItemInterface {
private:
    string  m_title;     //书名
    string  m_author;    //作者
    int m_pages;         //页数
public:
    Book(string p_author, string p_title, int p_pages);
    int getNumberOfPages();
    void accept(LibraryVisitor* visitor);
};
class LibraryVisitor {
public:
```

```
    (2)    ;
    (3)    ;
    virtual void printSum() = 0;
};
class LibrarySumPrintVisitor : public LibraryVisitor {    //打印总页数
private:
    int sum;
public:
    LibrarySumPrintVisitor();
    void visit(Book* p_book);
    void visit(Article* p_article);
    void printSum();
};
// visitor.cpp
int Article::getNumberOfPages(){
return m_end_page - m_start_page;
}
void Article::accept(LibraryVisitor* visitor) {    (4)    ;}
Book: :Book(string p_author, string p_title, int p_pages ) {
    m_title = p_title;
    m_author = p_author;
    m_pages = p_pages;
}
int Book::getNumberOfPages(){    return m_pages;  }
void Book::accept(LibraryVisitor* visitor){    (5)    ; }
//其余代码省略
```

解析：

本题主要考查设计模式中的访问者模式的访问。访问者模式就是表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。访问者模式把数据结构和作用于结构上的操作之间的耦合解脱开，使得操作集合可以相对自由地演化。该模式的目的是要把处理从数据结构分离出来。访问者模式让增加新的操作很容易，因为增加新的操作就意味着增加一个新的访问者。访问者模式将有关的行为集中到一个访问者对象中。

以上图 Visitor 模式类图为例，说明各个类的功能。

LibraryVisitor 类：抽象访问者角色，声明了两个访问操作，分别访问 Book 和 Article。

LibrarySumPrintVisitor 类：具体访问者角色，实现 LibraryVisitor 所声明的接口，也就是访问 Book 和 Article 的操作。

LibraryItemInterface 类：抽象被访问者角色，声明了一个接受操作，接受一个访问者对象。

Book 类和 Article 类：具体被访问者角色，实现了 LibraryItemInterface 类的接受操作。

根据上面的分析，LibraryItemInterface 类中应声明一个接受操作，用来接受访问者对象，再根据下面具体类的实现，可以推断空(1)中应填入 virtual void accept(LibraryVisitor* visitor)=0。

LibraryVisitor 类应声明对 Book 类和 Article 类的访问操作，再结合 LibrarySumPrintVisitor

类中的具体实现，可以推断空(2)中应填入 virtual void visit(Book* p_book)=0，空(3)中应填入 virtual void visit(Article* p_article)=0。

空(4)和空(5)是对 Book 类和 Article 类的 accept 的具体实现，因为对各个对象具体的访问实现是定义在 LibraryVisitor 类的 visit 方法中的，所以这两处应该完成对 visit 方法的调用，空(4)和空(5)应都填入(LibraryVisitor*)visitor->visit(this)。

答案：

(1) virtual void accept(LibraryVisitor* visitor)=0。

(2) virtual void visit(Book* p_book)=0。

(3) virtual void visit(Article* p_article)=0。

(4) (LibraryVisitor*)visitor->visit(this)。

(5) (LibraryVisitor*)visitor->visit(this)。

6.1.3　同步练习

1. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2009 年 11 月试题五)

【说明】

现欲构造一文件/目录树，采用组合(Composite)设计模式来设计，得到的类图如图 6-16 所示。

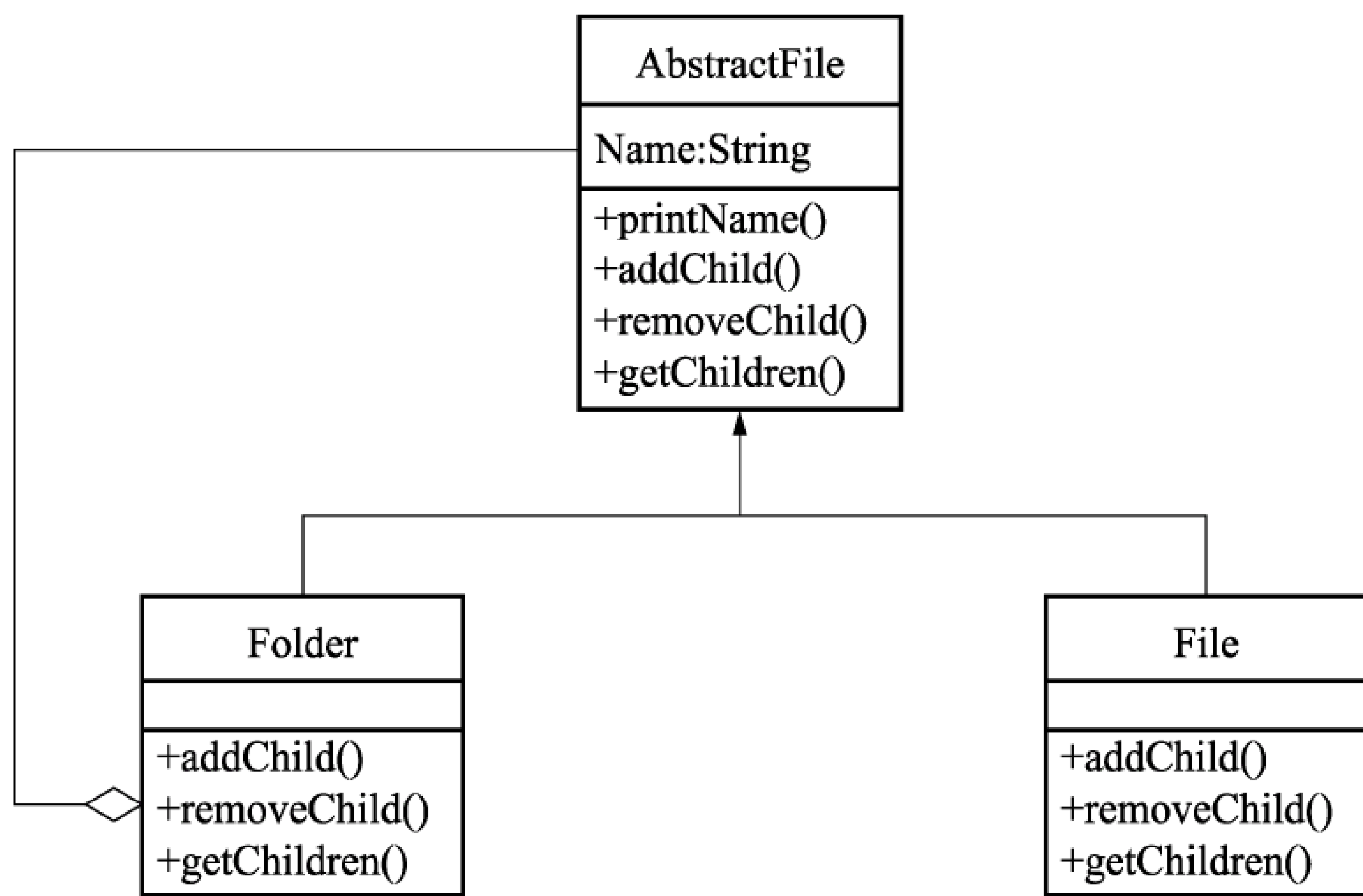


图 6-16　第 1 题类图

【C++程序】

```
#include <list>
#include <iostream>
#include <string>
using namespace std;
class AbstractFile
```

```
{
protected:
    string name;   //文件或目录名称
public:
    void printName(){cout<<name;}   //打印文件或目录名称
    virtual void addChild(AbstractFile *file)=0;  //给一个目录增加子目录或文件
    virtual void removeChild(AbstractFile *file)=0;  //删除一个目录的子目录或
                                                     //文件
    virtual list<AbstractFile*> *getChildren()=0; //获得一个目录的子目录或文件
};
class file:public AbstractFile
{
public:
   File(string name) { (1) = name;}
    void addChild(AbstractFile *file){return;}
    void removeChild(AbstractFile *file) {return;}
     (2)  getChildren(){return (3) ;}
};
class Folder:public AbstractFile
{
private:
    list <AbstractFile*> childList:  //存储子目录或文件
public:
    Folder(string name){ (4) =name;}
    void addChild(AbstractFile*file){childList.push_back(file);}
    void removeChild(AbstractFile*file){childList.remove(file);}
    list<AbstractFile*>*getChildren(){return (5) ;}
};
void main()
{
    //构造一个树型的文件/目录结构
    AbstractFile *rootFolder=new Folder("c:\\ ");
    AbstractFile*compositeFolder=new Folder("composite");
    AbstractFile *windowsFolder=new Folder("windows");
    AbstractFile*file=new File("TestComposite.java");
    rootFolder->addChild(compositeFolder);
    rootFolder->addChild(windowsFolder);
    compositeFolder->addChild(file);
}
```

2. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2009 年 5 月试题六)

【说明】

现欲实现一个图像浏览系统，要求该系统能够显示 BMP、JPEG 和 GIF 三种格式的文件，并且能够在 Windows 和 Linux 两种操作系统上运行。系统首先将 BMP、JPEG 和 GIF 三种格式的文件解析为像素矩阵，然后将像素矩阵显示在屏幕上。系统需具有较好的扩展性以支持新的文件格式和操作系统。为满足上述需求并减少所需生成的子类数目，采用桥

接(Bridge)设计模式进行设计，所得类图如图 6-17 所示。

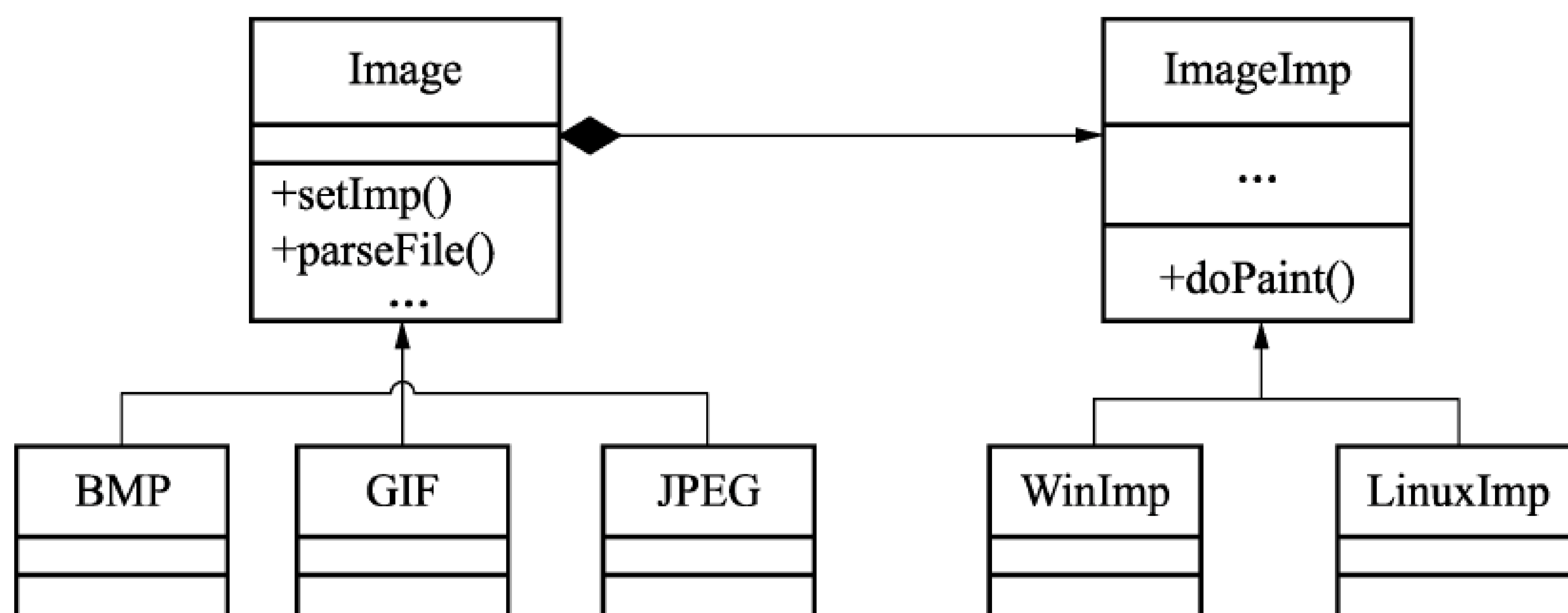


图 6-17　第 2 题类图

采用该设计模式的原因在于：系统解析 BMP、GIF 与 JPEG 文件的代码仅与文件格式相关，而在屏幕上显示像素矩阵的代码则仅与操作系统相关。

【C++程序】

```
class Matrix
{   //各种格式的文件最终都被转化为像素矩阵
    //此处代码省略
};
class ImageImp
{
public:
    virtual void doPaint(Matrix m) = 0;    //显示像素矩阵 m
};
class WinImp : public ImageImp
{
public:
    void doPaint(Matrix m){ /*调用 Windows 系统的绘制函数绘制像素矩阵*/ }
};
class LinuxImp : public ImageImp
{
public:
    void doPaint(Matrix m){ /*调用 Linux 系统的绘制函数绘制像素矩阵*/  }
};
class Image
{
public:
      void setImp(ImageImp *imp){ (1) = imp;}
      virtual void parseFile(string fileName) = 0;
protected:
      (2) *imp;
};
class BMP : public Image
{
public:
    void parseFile(string fileName){
```

```
            //此处解析 BMP 文件并获得一个像素矩阵对象 m
            (3) ;  //显示像素矩阵 m
        }
};
class GIF : public Image
{
    //此处代码省略
};
class JPEG : public Image
{
    //此处代码省略
};
void main()
{
    //在 Windows 操作系统上查看 demo.bmp 图像文件
    Image *image1 =  (4) ;
    ImageImp *imageImp1 = (5) ;
       (6)  ;
    image1->parseFile("demo.bmp");
}
```

现假设该系统需要支持 10 种格式的图像文件和 5 种操作系统，不考虑类 Matrix，若采用桥接设计模式，则至少需要设计 (7) 个类。

3. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2008 年 12 月试题六)

【说明】

已知某类库开发商提供了一套类库，类库中定义了 Application 类和 Document 类，它们之间的关系如图 6-18 所示。其中，Application 类表示应用程序自身，而 Document 类则表示应用程序打开的文档。Application 类负责打开一个已有的以外部形式存储的文档，如一个文件，一旦从该文件中读出信息后，它就由一个 Document 对象表示。

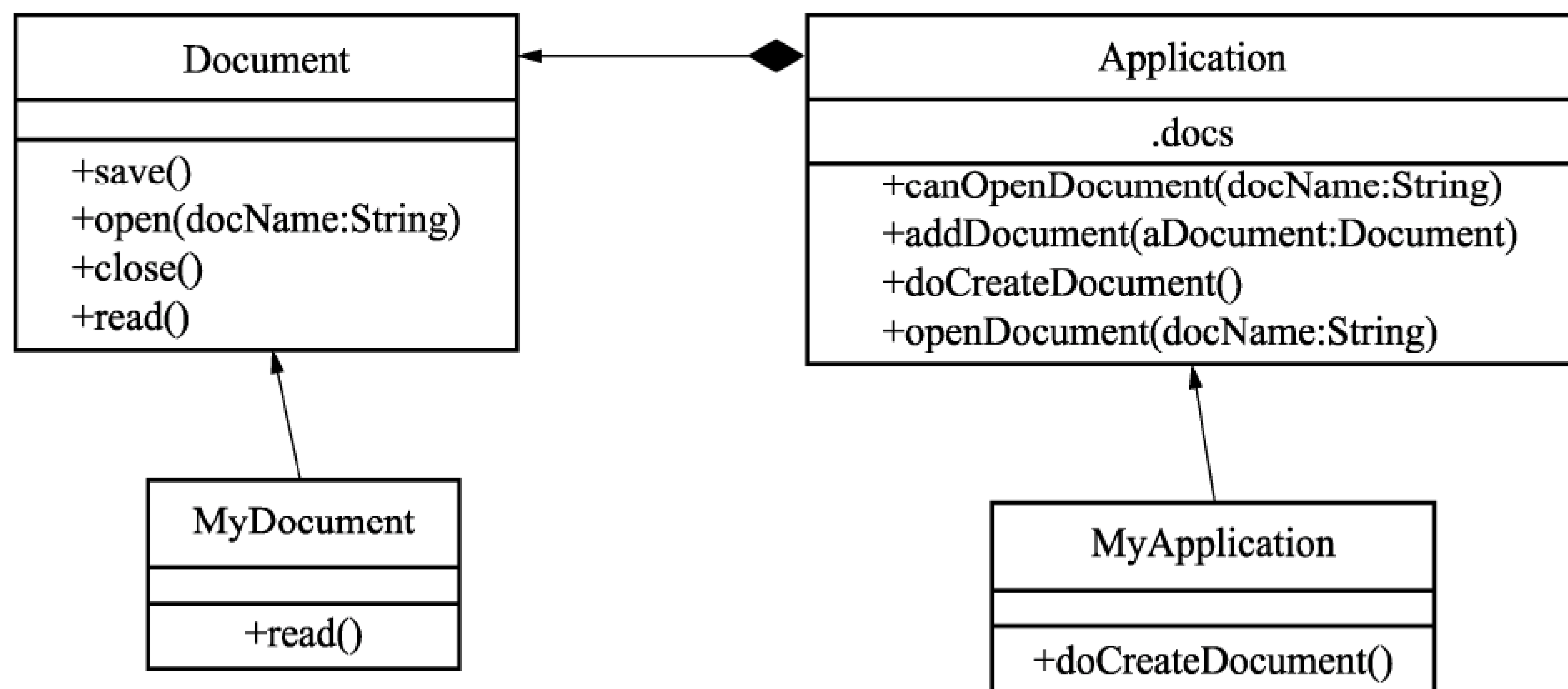


图 6-18　Application 与 Document 的关系

当开发一个具体的应用程序时，开发者需要分别创建自己的 Application 和 Document 子

类，如图 6-18 中的类 MyApplication 和类 MyDocument，并分别实现 Application 和 Document 类中的某些方法。

已知 Application 类中的 openDocument 方法采用了模板方法(Template Method)设计模式，该方法定义了打开文档的每一个主要步骤，具体如下。

(1) 首先检查文档是否能够被打开，若不能打开，则给出出错信息并返回。

(2) 创建文档对象。

(3) 通过文档对象打开文档。

(4) 通过文档对象读取文档信息。

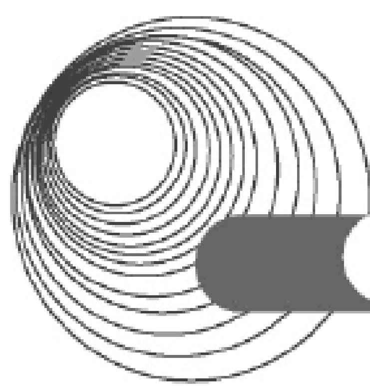
(5) 将文档对象加入到 Application 的文档对象集合中。

【C++程序】

```
#include <iostream>
#include <vector>
using namespace std;     // 使用全局的命名域方式

class Document
{
public:
        void save(){ /*存储文档数据，此处代码省略*/ }
        void open(string docName){ /*打开文档，此处代码省略*/ }
        void close(){ /*关闭文档，此处代码省略*/ }
        virtual void read(string docName) = 0;
};
class Application{
private:
    vector <  (1)  > docs; /*文档对象集合*/

public:
    bool canOpenDocument(string docName){
/*判断是否可以打开指定文档，返回真值表示可以打开，返回假值表示不可打开，此处代码省略*/
}
    void addDocument(Document * aDocument){
/*将文档对象添加到文档对象集合中*/
    docs.push_back(  (2)  );
}
    virtual Document * doCreateDocument() = 0; /*创建一个文档对象*/
    void openDocument(string docName){ /*打开文档*/
        if ( (3) ){
        cout << "文档无法打开！" << endl;
        return;
        }
          (4)  adoc =  (5)  ;
          (6)  ;
          (7)  ;
          (8)  ;
    }
};
```



4. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2008 年 5 月试题六)

【说明】

已知某企业欲开发一家用电器遥控系统，即用户使用一个遥控器就可控制某些家用电器的开与关。遥控器如图 6-19 所示。该遥控器共有 4 个按钮，编号分别是 0~3，按钮 0 和按钮 2 能够遥控打开电器 1 和电器 2，按钮 1 和按钮 3 则能遥控关闭电器 1 和电器 2。由于遥控系统需要支持形式多样的电器，因此该遥控系统的设计要求具有较高的扩展性。现假设需要控制客厅电视和卧室电灯，对该遥控系统进行设计所得类图如图 6-20 所示。

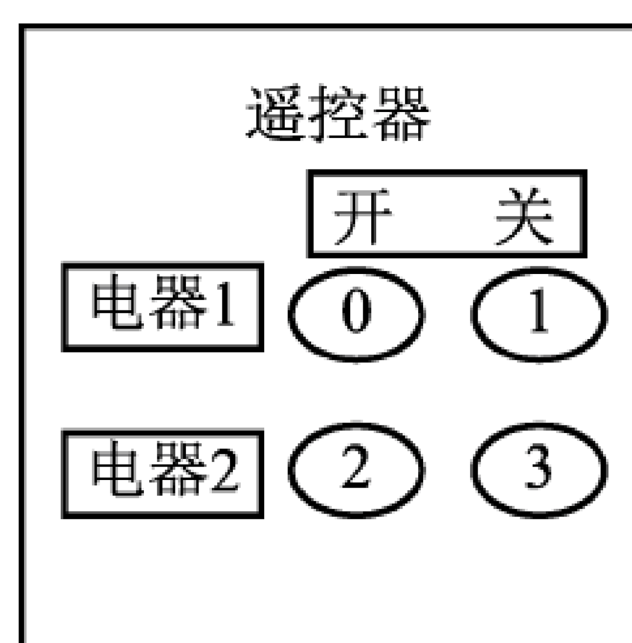


图 6-19 遥控器

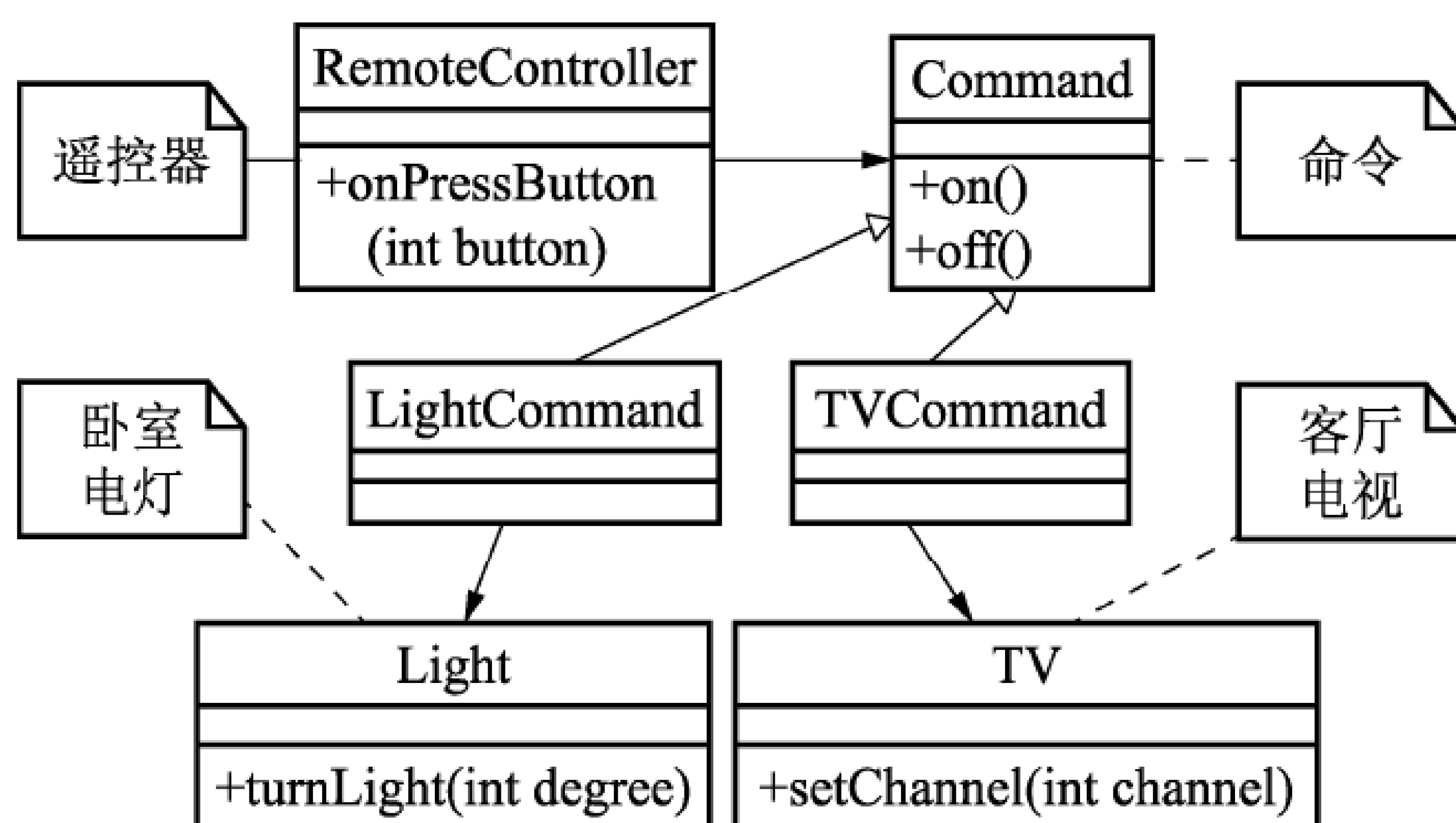


图 6-20 设计类图

在图 6-20 中，类 RomoteController 的方法 onPressButton(int button)表示当遥控器按键按下时调用的方法，参数为按键的编号；Command 接口中 on 和 off 方法分别用于控制电器的开与关；Light 中 turnLight(int degree)方法用于调整电灯光的强弱，参数 degree 值为 0 时表示关灯，值为 100 时表示开灯，并且将灯光亮度调整到最大；TV 中 setChannel(int channel)方法表示设置电视播放的频道，参数 channel 值为 0 时表示关闭电视，为 1 时表示开机并将频道切换为第 1 频道。

【C++程序】

```
class Light{ //电灯类
public:
    void turnLight(int degree){ //调整灯光亮度，0 表示关灯，100 表示亮度最大 }
};
class TV{ //电视机类
public:
    void setChannel(int channel){//调整电视频道，0 表示关机，1 表示开机并切换到 1 频道
};
class Command{ //抽象命令类
public:
    virtual void on()=0;
    virtual void off()=0;
};
class RemoteController{ //遥控器类
protected:
```

```
    Command *commands[4]; //遥控器有 4 个按钮，按照编号分别对应 4 个 Command 对象
public:
    void onPressButton(int button){ //按钮被按下时执行命令对象中的命令
            if(button % 2 == 0)commands[button]->on();
            else commands[button]->off();
    }
    void setCommand(int button,Command * command){
            (1) = command; //设置每个按钮对应的命令对象
    }
};
class LightCommand : public Command{ //电灯命令类
protected: Light *light; //指向要控制的电灯对象
public:
    void on(){light->turnLight(100);};
    void off(){light-> (2) ;};
    LightCommand(Light * light){this->light = light;};
};
class TVCommand : public Command{ //电视机命令类
protected: TV * tv; //指向要控制的电视机对象
public:
    void on(){tv-> (3) ;};
    void off(){tv->setChannel(0);};
    TVCommand(TV * tv){ this->tv = tv; };
};
void main()
{
    Light light; TV tv; //创建电灯和电视对象
    LightCommand lightCommand(&light);
    TVCommand tvCommand(&tv);
    RemoteController remoteController;
    remoteController.setCommand(0, (4) ); //设置按钮 0 的命令对象

    …//此处省略设置按钮 1、按钮 2 和按钮 3 的命令对象代码
}
```

本题中，应用命令模式能够有效地让类 (5) 和类 (6) 、类 (7) 之间的耦合性降至最小。

5. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2007 年 11 月试题六)

【说明】

已知某企业的采购审批是分级进行的，即根据采购金额的不同由不同层次的主管人员来审批，主任可以审批 5 万元以下(不包括 5 万元)的采购单，副董事长可以审批 5 万~10 万元(不包括 10 万元)的采购单，董事长可以审批 10 万~50 万元(不包括 50 万元)的采购单，50 万元及以上的采购单就需要开会讨论决定。

采用责任链设计模式(Chain of Responsibility)对上述过程进行设计后得到的类图如图 6-21 所示。

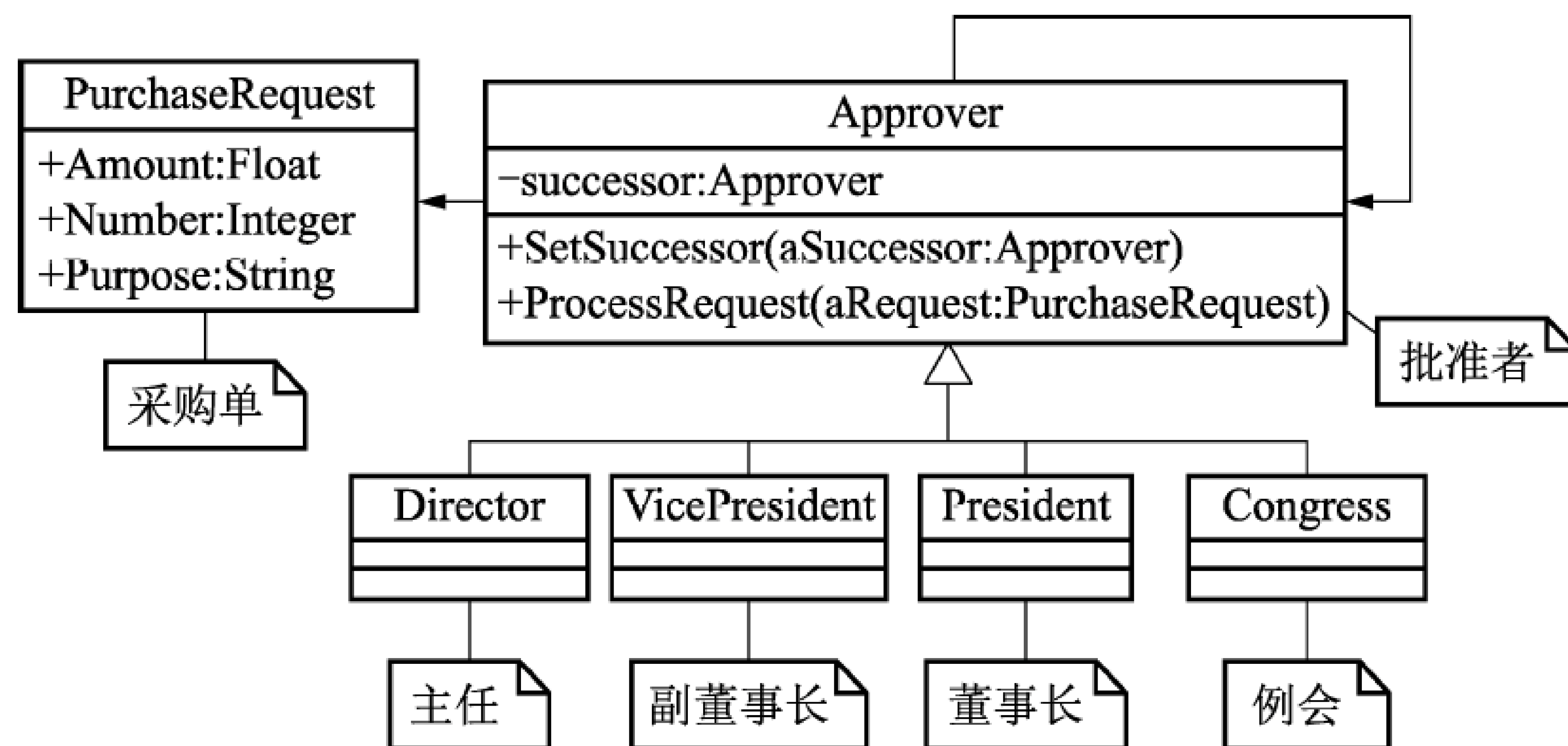


图 6-21　设计类图

【C++程序】

```
#include <string>
#include <iostream>
using namespace std;
class PurchaseRequest
{
public:
       double Amount;    // 一个采购的金额
       int Number;       // 采购单编号
       string Purpose;   // 采购目的
};
class Approver           // 审批者类
{
public:
    Approver(){ successor = NULL;   }
    virtual void ProcessRequest(PurchaseRequest aRequest){
        if (successor != NULL){  successor->__(1)__;}
    }
    void SetSuccessor(Approver *aSuccessor){ successor = aSuccessor; }
private:
    __(2)__successor;
};
class Congress : public Approver {
public:
       void ProcessRequest(PurchaseRequest aRequest){
        if(aRequest.Amount >= 500000){ /* 决定是否审批的代码省略 */    }
        else __(3)__ProcessRequest(aRequest);
    }
};
class Director : public Approver
{
public:
       void ProcessRequest(PurchaseRequest aRequest){  /* 此处代码省略 */}
};
class President : public Approver
{
```

```
public:
    void ProcessRequest(PurchaseRequest aRequest){ /* 此处代码省略 */ }
};
class VicePresident : public Approver
{
public:
    void ProcessRequest(PurchaseRequest aRequest){ /* 此处代码省略 */  }
};
void main()
{
    Congress Meeting; VicePresident Sam; Director Larry; President Tammy;
    // 构造责任链
    Meeting.SetSuccessor(NULL);Sam.SetSuccessor(__(4)__);
    Tammy.SetSuccessor(__(5)__);Larry.SetSuccessor(__(6)__);

    PurchaseRequest aRequest;          // 构造一采购审批请求
    cin >> aRequest.Amount;            // 输入采购请求的金额
    __(7)__.ProcessRequest(aRequest);  // 开始审批
    return;
}
```

6. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2007 年 5 月试题六)

【说明】

某游戏公司现欲开发一款面向儿童的模拟游戏，该游戏主要模拟现实世界中各种鸭子的发声特征、飞行特征和外观特征。游戏需要模拟的鸭子种类及其特征如表 6-5 所示。

表 6-5　游戏需要模拟的鸭子种类及其特征

鸭子种类	发声特征	飞行特征	外观特征
灰鸭(MallardDuck)	发出“嘎嘎”声(Quack)	用翅膀飞行(FlyWithWings)	灰色羽毛
红头鸭(RedHeadDuck)	发出“嘎嘎”声(Quack)	用翅膀飞行(FlyWithWings)	灰色羽毛、头部红色
棉花鸭(CottonDuck)	不发声(QuackNoWay)	不能飞行(FlyNoWay)	白色
橡皮鸭(RubberDuck)	发出橡皮与空气摩擦的声音(Squeak)	不能飞行(FlyNoWay)	黑白橡皮色

为支持将来能够模拟更多种类鸭子的特征，采用策略设计模式(Strategy)设计的类图如图 6-22 所示。

其中，Duck 为抽象类，描述了抽象的鸭子，而类 RubberDuck、MallardDuck、CottonDuck 和 RedHeadDuck 分别描述具体的鸭子种类，方法 fly()、quack()和 display()分别表示不同种类的鸭子都具有飞行特征、发声特征和外观特征；类 FlyBehavior 与 QuackBehavior 为抽象类，分别用于表示抽象的飞行行为与发声行为；类 FlyNoWay 与 FlyWithWings 分别描述不能飞行的行为和用翅膀飞行的行为；类 Quack、Squeak 与 QuackNoWay 分别描述发出“嘎嘎”声的行为、发出橡皮与空气摩擦声的行为与不发声的行为。请填补以下代码中的空缺。

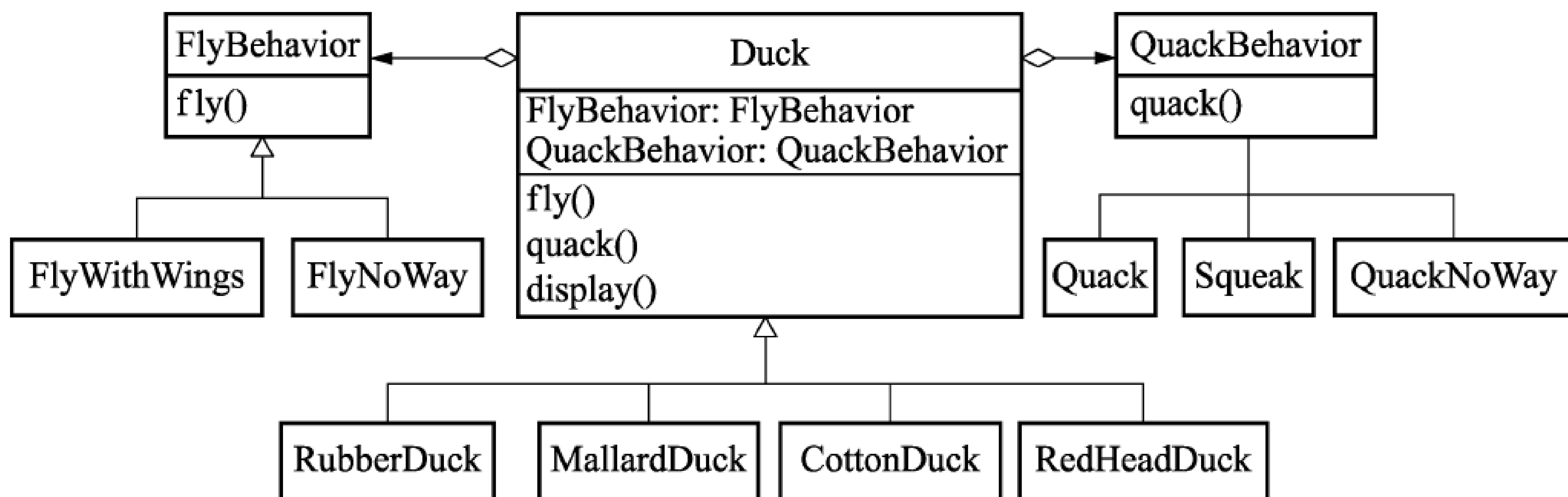


图 6-22　第 6 题类图

【C++程序】

```
#include <iostream>
using namespace (1) ;
class FlyBehavior
{
    public: (2)  fly( )=0;
};
class QuackBehavior
{
    public: (3) quack( )=0;
};

class FlyWithWings:public FlyBehavior
{
    public: void fly( ){cout<<"使用翅膀飞行！"<<endl;}
};
class FlyNoWay:public FlyBehavior
{
    public: void fly( ){cout<<"不能飞行！"<<endl;}
};
class Quack:public QuackBehavior
{
    public: void quack( ){cout<<"发出\'嘎嘎'\声！"<<endl;}
};
class Squeak:public QuackBehavior
{
    public: void quack( ){cout<<"发出空气与橡皮摩擦声！"<<endl;}
};
class QuackNoWay:public QuackBehavior
{
    public: void quack( ){cout<<"不能发声！"<<endl;}
};
Class Duck
{
protected:
    FlyBehavior * (4) ;
    QuackBehavior *  (5) ;
public:
```

```
    void fly(){  (6) ;  }
    void quack() {  (7) ; };
    virtual void display()=0;
};
class RubberDuck: public Duck
{
public:
    RubberDuck( )
    {
        flyBehavior=new (8) ;
        quackBehavior=new (9) ;
    }
    ~RubberDuck( )
    {
        if (!flyBehavior) delete flyBehavior;
        if (!quackBehavior) delete quackBehavior;
    }
    void display( ){/*此处省略显示橡皮鸭的代码*/
};
//其他代码省略
```

7. 阅读以下说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2006 年 11 月试题六)

【说明】

传输门是传输系统中的重要装置。传输门具有 Open(打开)、Closed(关闭)、Opening(正在打开)、StayOpen(保持打开)、Closing(正在关闭)5 种状态。触发传输门状态转换的事件有 click、complete 和 timeout 3 种。事件与其相应的状态转换如图 6-23 所示。

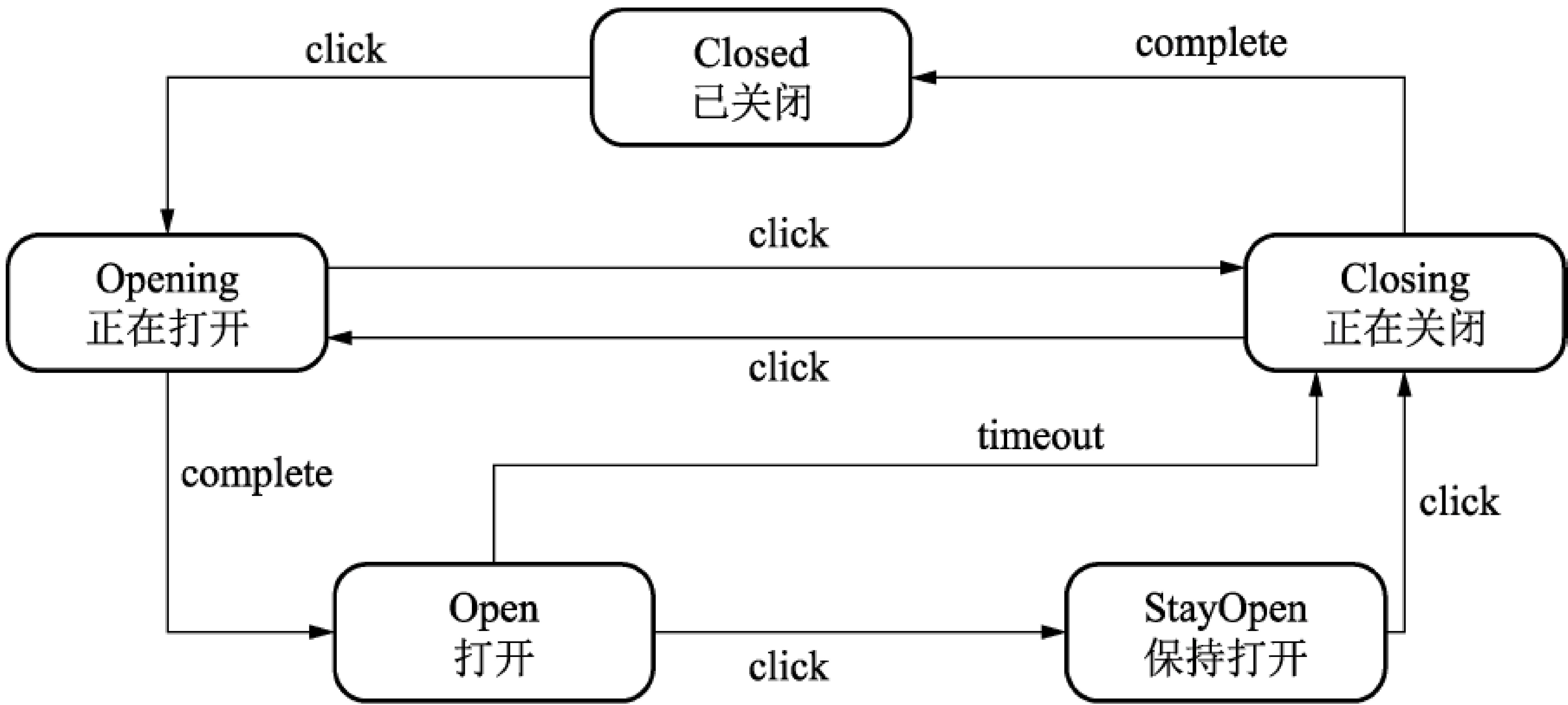


图 6-23 传输门响应事件与其状态转换图

下面的 C++程序 1 与 C++程序 2 分别用两种不同的设计思路对传输门进行状态模拟，请填补代码中的空缺。

【C++程序 1】

```
const int CLOSED = 1; const int OPENING = 2;
const int OPEN = 3; const int CLOSING = 4;
```

```
const int STAYOPEN = 5; //定义状态变量，用不同整数表示不同状态
class Door
{
   private:
      int state; //传输门当前状态
      void setState(int state){ this->state = state; } //设置当前状态
   public:
      Door():state(CLOSED){};
      void getState(){  //根据当前状态输出相应的字符串
      switch(state)
      {
        case OPENING: cout <<"OPENING" << endl; break;
        case CLOSED: cout << "CLOSED" << endl; break;
        case OPEN: cout << "OPEN" << endl; break;
        case CLOSING: cout << "CLOSING" << endl; break;
        case STAYOPEN: cout << "STAYOPEN" << endl; break;
      }
   }
     void click()    //发生 click 事件时进行状态转换
     {
        if (  (1)  ) setState(OPENING);
        else if (  (2)  ) setState(CLOSING);
        else if (  (3)  ) setState(STAYOPEN);
     }
     void timeout()    //发生 timeout 事件时进行状态转换
     {
        if (state == OPEN) setState(CLOSING);
     }
     void complete()    //发生 complete 事件时进行状态转换
     {
        if (state == OPENING) setState(OPEN);
        else if (state == CLOSING) setState(CLOSED);
     }
};
int main()
{
      Door aDoor;
      aDoor.getState(); aDoor.click(); aDoor.getState();
      aDoor.complete();
      aDoor.getState(); aDoor.click(); aDoor.getState();
      aDoor.click();
      aDoor.getState(); return 0;
}
```

【C++程序 2】

```
class Door
{
   public:
       DoorState *CLOSED, *OPENING, *OPEN, *CLOSING, *STAYOPEN, *state;
```

```
         Door();
         virtual ~Door()
         { …   //释放申请的内存，此处代码省略};
         void setState(DoorState *state) { this->state = state; }
         void getState(){
        // 此处代码省略，本方法输出状态字符串
        // 例如，当前状态为 CLOSED 时，输出字符串为"CLOSED"
         };
        void click();
        void timeout();
        void complete();
    };
    Door::Door()
{
        CLOSED = new DoorClosed(this);OPENING = new DoorOpening(this);
        OPEN = new DoorOpen(this);CLOSING = new DoorClosing(this);
        STAYOPEN = new DoorStayOpen(this);
        state = CLOSED;
        //设置当前传输门的状态为 CLOSED
}
    void Door::click(){ (4) ;}
    void Door::timeout(){ (5) ; }
    void Door::complete(){ (6) ; }
class DoorState  //定义一个抽象的状态，它是所有状态类的基类
{
    protected: Door *door;
    public:
        DoorState(Door *door) { this->door = door; }
        virtual ~DoorState(void);
        virtual void click() {}
        virtual void complete() {}
        virtual void timeout() {}
  };
class DoorClosed:public DoorState    //定义一个基本的 Closed 状态
{
    public:
        DoorClosed(Door *door):DoorState(door) {}
        virtual~DoorClosed (){}
        void click();
  };
    void DoorClosed::click() { (7) ; }

// 其他状态类的定义与实现代码省略
int main()
{
      Door aDoor;
      aDoor.getState();  aDoor.click();  aDoor.getState();
      aDoor.complete();
      aDoor.getState();  aDoor.timeout();  aDoor.getState();  return 0;
```

```
}
```

8. 阅读下列说明、图和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2006 年 5 月试题六)

【说明】

某订单管理系统的部分 UML 类图如图 6-24 所示。

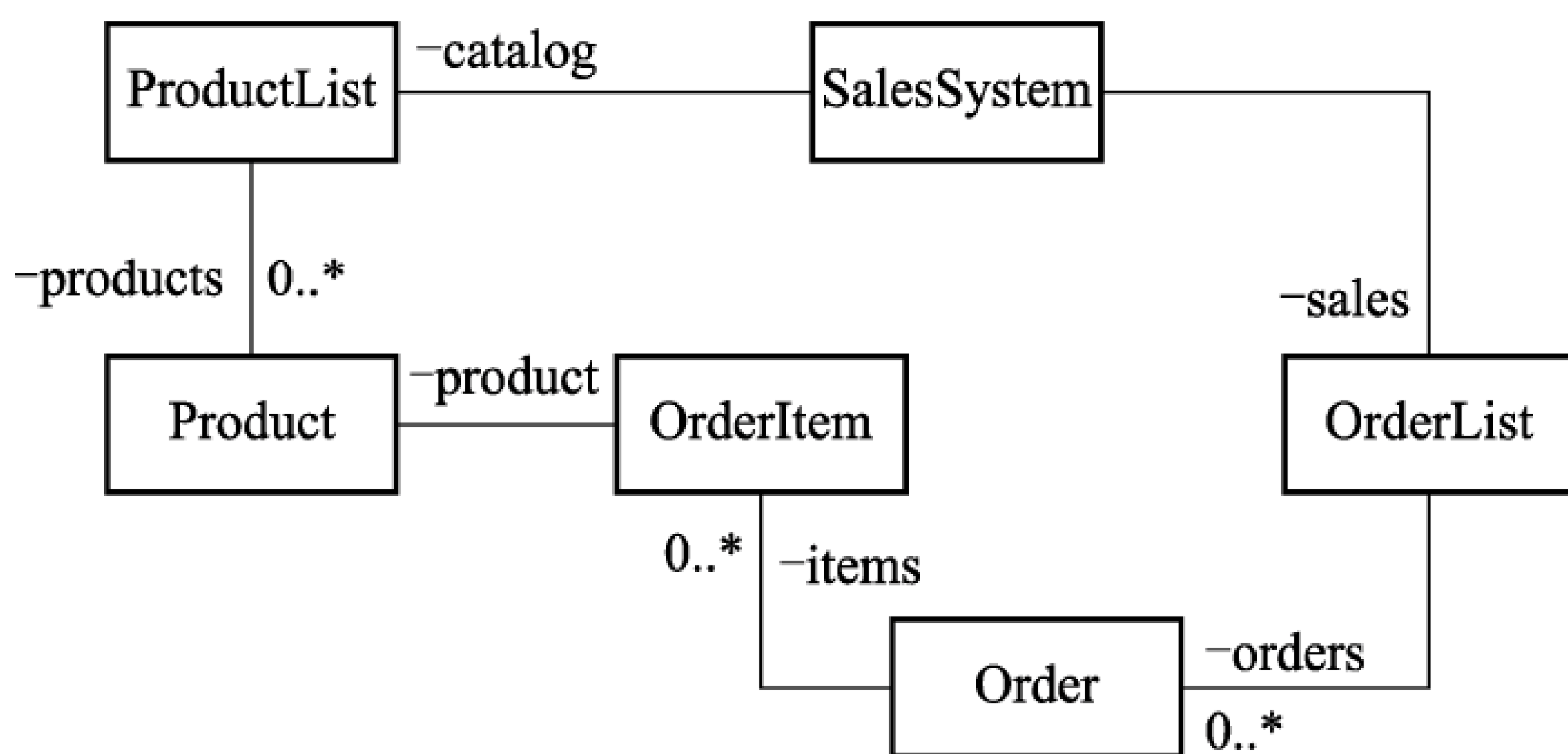


图 6-24　UML 类图

在图 6-24 中，Product 表示产品，ProductList 表示产品目录，Order 表示产品订单，OrderItem 表示产品订单中的一个条目，OrderList 表示订单列表，SalesSystem 提供订单管理系统的操作接口。

请完善类 Order 的成员函数 getOrderedAmount()和类 SalesSystem 的 statistic()方法，各个类的属性及部分方法定义参见下面的 C++代码。

【C++程序】

```
class Product{                     //产品类
private:
    string pid;                    //产品识别码
    string description;            //产品描述
    double price;                  //产品单价
public:
    void setProductPrice(double price);       //设置产品单价
    string getProductId();                    //获取产品识别码
    string getProductDescription ();          //获取产品描述
    double getProductPrice();                 //获取产品单价
    //其他成员省略
};

class ProductList                             //产品列表类
{
private:
    vector <Product> products;
public:
    ProductList();
    Product getProductByIndex(int i);         //获得产品列表中的第 i 件产品
    void addProduct(Product t);               //在产品列表中加入一件产品
    Product * getProductByID(string pid);     //获得识别码为 pid 的产品指针
```

```
    unsigned int getProductAmount();        //获得产品列表中的产品数量
};

class OrderItem
{                                   //订单条目类
private:
    Product *productPtr;            //指向被订购产品的指针
    int quantity;                   //订购数量
public:
    OrderItem (Product *,int);
    Product * getProductptr();      //获取指向被订购产品的指针
    int getQuantity();              //获取被订购产品的数量
};

class Order{                        //订单类
private:
    unsigned int orderid;           //订单识别号
    vector<OrderItem> items;        //订单内容(订单项)
public:
    Order(unsigned int orderid);
    //获得识别码为 tid 的产品在当前订单中被订购的数量
    int getOrderedAmount(string tid);
    void additem(Product *productPtr,unsigned int n);//在订单中增加一个订单项
    //其他成员省略
};

class OrderList{                    //订单列表类
private:
    vector<Order> orders;
public:
    OrderList();
    //Begin()返回指向订单列表第一个元素的迭代器(指针)
    virtual vector<Order>::iterator OrderList::Begin();
    //End()返回指向订单列表最后一个元素之后的迭代器(指向一个不存在的元素)
    virtual vector<Order>::iterator OrderList::End();
    void addOrder(Order t);         //在订单列表中加入一份订单
    //其他成员省略
};

class SalesSystem
{
private:
    ProductList catalog;    //产品目录
    OrderList sales;        //订单列表
public:
    SalesSystem();
    void statistic();       //统计所有产品的订购情况
    //其他成员省略
```

```
};

//在订单中查找识别码为 tid 的产品的订购数量，若该产品没有被订购，则返回 0
int Order::getOrderedAmount(string tid)
{
    for(int k = 0; k < items.size(); k++)
{
        if( (1)  == tid)
            return  (2) ;
     }
    return 0;
}

// 方法 statistic()依次统计产品目录中每个产品的订购总量，并打印输出
// 每个产品的识别码、描述、订购总量和订购金额
void SalesSystem::statistic()
{
    unsigned int k, t, ordered_qty = 0;
    vector<Order>::iterator it;
    Product p;
    cout << "产品识别码\t 描述\t\t 订购数量\t 金额" << endl;

    for(k = 0; k < catalog.getProductAmount(); k++){//搜索产品列表
        p =  (3) ;          //从产品列表取得一件产品信息存入变量 p
        ordered_qty = 0;
        //通过迭代器变量 it 搜索订单列表中的每一份订单
        for (it = sales.Begin(); it <  (4) ; it++) {
            //根据产品识别码获得产品 p 在当前订单中被订购的数量
            t =  (5) (p.getProductId());
            ordered_qty += t;
        }
    cout<<p.getProductId()<<"\t\t"<<p.getProductDescription()<<"\t\t";
    cout<<ordered_qty<<"\t\t"<<p.getProductPrice()*ordered_qty<<endl;
    }
}
```

9. 阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。(2016 年 5 月试题五)

【说明】

某软件系统中，已设计并实现了用于显示地址信息的类 Address(如图 6-25 所示)，现要求提供基于 Dutch 语言的地址信息显示接口。为了实现该要求并考虑到以后可能还会出现新的语言的接口，决定采用适配器(Adapter)模式实现该要求，得到如图 6-25 所示的类图。

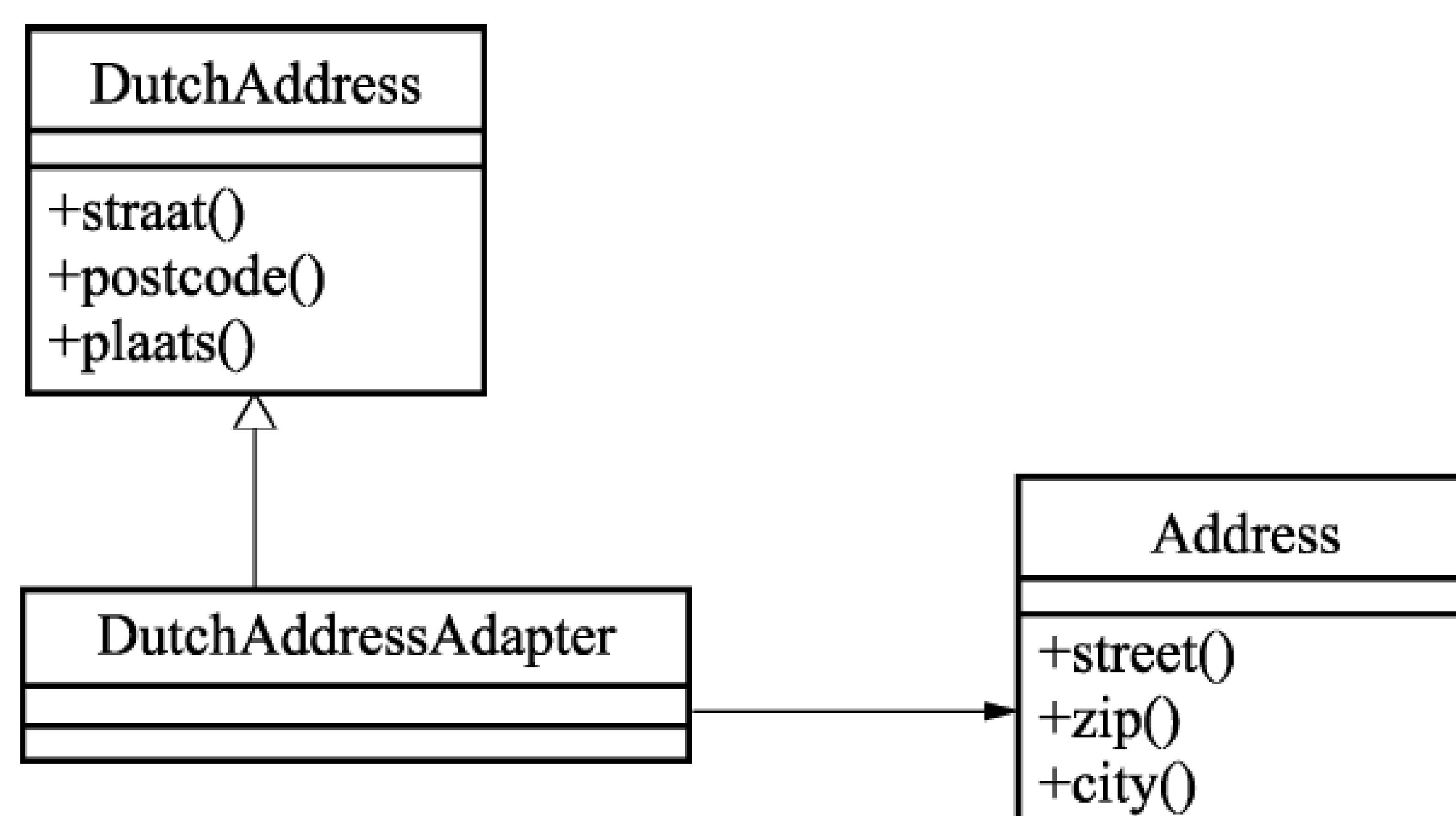


图 6-25　适配器模式类图

【C++代码】

```
#include <iostream>
using namespace std;
class Address{
public:
void street(){ /*实现代码省略*/}
void zip(){ /*实现代码省略*/}
void city() { /*实现代码省略*/}
//其他成员省略
};
class DutchAddress {
public:
virtual void straat()=0;
virtual void postcode()=0;
virtual void plaats()=0;
//其他成员省略
};
class DutchAddressAdapter : public DutchAddress {
private:
(1);
public:
DutchAddressAdapter(Address *addr) {
address = addr;
}
void straat() {
(2);
}
void postcode(){
(3);
}
void plaats(){
(4);
}
//其他成员省略
};
void testDutch(DutchAddress *addr){
```

```
addr->straat();
addr->postcode();
addr->plaats();
}
int main(){
Address*addr = new Address();
(5);
cout<< "\n The DutchAddress\n"<< endl;
testDutch(addrAdapter);
return 0;
}
```

6.1.4 同步练习参考答案

1.

(1) this->name。 (2) list<AbstractFile*>*。 (3) null。

(4) this->name。 (5) &childList。

2.

(1) this->imp。 (2) ImageImp。 (3) imp->doPaint(m)。 (4) new BMP()。

(5) new WinImp()。 (6) image1->setImp(imageImp1)。 (7) 17。

3.

(1) Document*。 (2) aDocument。 (3) !canOpenDocument(docName)。

(4) Document*。 (5) doCreateDocument()。 (6) adoc->open(docName)。

(7) adoc->read(docName)。 (8) addDocument(adoc)。

4.

(1) commands[button]。 (2) turnLight(0)。 (3) setChannel(1)。

(4) &lightCommand。 (5) RemoteController。 (6) Light。 (7) TV。

5.

(1) ProcessRequest(aRequest)。 (2) Approver *。 (3) Approver::。

(4) &Tammy。 (5) &Meeting。 (6) &Sam。 (7) Larry。

6.

(1) std。 (2) virtual void。 (3) virtual void。 (4) flyBehavior。

(5) quackBehavior。 (6) flyBehavior->fly()。 (7) quackBehavior->quack()。

(8) FlyNoWay()。 (9) Squeak()。

7.

(1) state==CLOSED||state==CLOSING。

(2) state==OPENING||state==STAYOPEN。

(3) state==OPEN。

(4) state->click()。

(5) state->timeout()。

(6) state->complete()。

(7) door->setState(door->OPENING)。

8.

(1) (items[k].getProductptr())->getProductId()。

(2) items[k].getQuantity()。

(3) catalog.getProductByIndex(k)。

(4) sales.End()。

(5) it->getOrderedAmount。

9.

(1) Address* address。

(2) address->street()。

(3) address->zip()。

(4) address->city()。

(5) DutchAddress *addr=new DutchAddressAdaptor(addr)。

6.2　Java 基础知识

6.2.1　考点辅导

6.2.1.1　基本概念

1. 应用领域

Java 目前主要应用于服务器端的企业级应用(Servlet、JSP)、手持设备(J2ME、K-Java、无线 Java)、普通网页(Applet)、普通应用程序。

2. 优点

(1) 跨平台(大部分平台上都有 Java 虚拟机)。许多平台(计算机+操作系统)上都有各自的 Java 虚拟机(Java VM)，Java 虚拟机不跨平台，要分别编写。编译生成的是中间代码，由统一的 Java 虚拟机指令组成。

(2) 代码可移动(与 HTML 相结合)。

(3) 完全面向对象。

(4) 编出来的程序不易出错(没有指针，内存垃圾自动回收，不会产生内存泄漏)。

此外，还有简单、安全、多线程等优点。

3. Java 与 C++的区别

(1) 完全面向对象：无全局变量、无结构和联合、自动回收内存垃圾。

(2) 没有指针。

(3) 没有多继承。

(4) 解释执行。

6.2.1.2 基本语法

1. 注释

与 C 语言相同。多行注释：/*…*/。单行注释：//。

2. 基本数据类型

与 C 语言相同：char(8 bit)、short(16 bit)、int(32 bit)、long(64 bit)、float(32 bit)、double (64 bit)。

不同：byte(8 bit)，boolean(boolean 类型的变量取值为 true 或者 false)。

3. 常量

与 C 语言不同，使用 final 关键字，C++语言则是 const，如 final float pi = 3.14f、final byte c = 12。浮点常数后面要加“f”，如 12.7f、1.02f。

4. 运算符

与 C 语言相同的有算术运算符、赋值运算符、逻辑运算符、比较运算符、自增自减运算符、位运算符和移位运算符。

5. 类型转换

与 C 语言相同，如 int s = (int)4.7f。

6. 基本语句

与 C 语言相同：if、switch、while、do{…}while、for。

输出函数：System.out.println 与 C 语言的 printf 函数不同，各输出项目之间用“+”连接，如 System.out.println("Id=" + nNum)。

6.2.1.3 程序设计

1. 类和继承

1) 类

一个类是一些属性和方法的封装体，类的定义用关键字 class 声明，用关键字 public、protected、private 指定类的成员的存取控制属性：private(私有)成员只有类内部的方法才能访问，protected(保护)成员派生类和同一文件夹下的类可以访问，public(公有)成员可以从类的外部访问。默认是 public。这体现了面向对象的以下指导思想：尽量将类内部的细节隐藏起来，对类的属性的操作应该通过类的方法来进行。

另外，public 还可以用来修饰类，public 类能够被其他文件夹下的类访问，非 public 类只能被同一文件夹下的类访问。一个.java 文件中可以包含多个类，会被编译成多个.class 文件，但只能有一个 public 类，而且该类名要和文件名一样。

2) 继承

Java 中用关键字 extends 表示类间的继承关系。父类的公有属性和方法成为子类的属性和方法，子类如果有和父类的同名、同参数类型的方法，那么子类对象在调用该方法时，

调用的是子类的方法，亦即方法的重置。如果想要调用父类的同名方法，需要用 super 关键字(属性同理)。

子类的对象可以作为祖先类的对象使用，即所谓类的向上转换，反之则不行。具体表现在：可以用子类对象来对祖先类对象赋值，可以用子类对象作为实参去调用以父类对象为形参的函数。

2. 对象的引用本质

Java 中的对象实际上是对象的引用，本质上和 C 语言中的指针是一样的；但也和 C 语言指针不尽相同，例如，不能自增、自减，不能强制转换成其他类型。

例如：

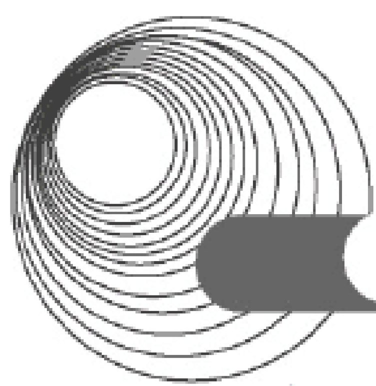
```
//ManKind.java 文件
public class ManKind {
    int sex;//默认是公有成员
    public void manOrWoman(){//公有方法
        if(sex ==0){//表示男人
            System.out.println("Man!");
        }else{//女人
            System.out.println("Woman!");
        }
    }
}
//Main.java 文件
public class Main {
    public static void main(String[] args){
        ManKind somePerson, somePerson2;
        //somePerson.sex = 1; //注意这里，出错，因 somePerson 尚未初始化
        somePerson = new ManKind();//初始化对象，注意后面的括号，不能省略
        somePerson.sex = 1;//初始化为 1，表示男人
        somePerson.manOrWoman();//输出“Woman”
        somePerson2 = somePerson;//将 somePerson 赋值给 somePerson2
        somePerson2.sex = 0;//修改 somePerson2 的 sex 属性为 0，即男人
        somePerson2.manOrWoman();//输出 somePerson2 的 sex，为“Man”
        somePerson.manOrWoman();//输出 somePerson，发现输出“Man”，可见通过
            //修改 somePerson2 的 sex 属性成功修改了 somePerson 的 sex 属性，亦即
            //somePerson 与 somePerson2 实际上是同一个对象
    }
}
```

3. 构造方法

构造方法就是类的对象生成时会被调用的方法。每个类至少有一个构造方法(Constructor)，也称构造函数。构造方法的名字和类名相同，没有任何返回类型。每个类都有一个默认的构造方法，但当用户自定义了构造方法后，默认的构造函数就不再有效了。

4. 重载

同一个类中的两个或两个以上方法，名字相同，而参数个数不同或参数类型不同，称



为重载。注意：不能有各方法名字和参数都一样，而仅仅返回值类型不同。

5. 静态属性和静态方法

静态属性和静态方法的声明用关键字 static 实现，一个类的静态属性只有一份，由所有该类的对象共享。不需要创建对象也能访问类的静态属性和方法，访问方式为“类名.静态属性或静态方法”，静态方法与对象无关，因此不能在静态方法中访问非静态属性和调用非静态方法。

6. this 和 super 关键字

这两个关键字颇为重要。this 代表当前对象，super 代表当前对象的父类的东西。

this 主要用途有以下两个。

(1) 一个构造函数调用另一个构造函数，对构造函数的调用必须是第一条语句。

(2) 将对象自身作为参数来调用一个函数。

super 的用途如下：在子类中调用父类的同名方法，或在子类的构造函数中调用父类的构造函数，此时亦必须是第一条语句。

7. 多态

所谓多态，是指通过基类对象调用一个基类和派生类都有的方法时，在运行时才能确定到底调用的是基类的方法还是派生类的方法。多态的好处是增加了程序的可扩展性。多态是通过动态联编实现的，即编译时不确定，程序运行时才确定调用哪个函数。

8. 抽象类与接口

1) 抽象类

抽象类通过关键字 abstract 实现，抽象类的目的是定义一个框架，规定某些类必须具有的一些共性。

包含抽象方法的类一定是抽象类，所谓抽象方法是指没有函数体的方法。

抽象类的直接派生类必须实现其抽象方法；抽象类只能用于继承，不能创建对象。

2) 接口(Interface)

接口用关键字 interface 声明，只能用于继承。注意：此时关键字为 implements(实现)。

接口用于替代多继承的概念，能实现多继承的部分特点，又避免了多继承的混乱，还能起到规定程序框架的作用。注意：接口也可以用于多态。

直接继承了接口的类，必须实现接口中的抽象方法；间接的则可以实现，也可以不实现。

3) 抽象类与接口的异同

接口和抽象类都不能创建对象。

抽象类不能参与多继承，抽象类可以有非静态的成员变量，可以有非抽象方法；接口可以参与多继承，所有属性都是静态常量，所有方法都是 public 抽象方法。

9. 异常处理

1) 异常概念

异常，即出错，比如 0 作为除数、找不到类、打开文件错误、数组越界等。异常如果不进行处理，那么程序运行就会结束；如果进行处理，那么会在执行完异常处理代码后继

续运行。

Java 中所有异常类均继承自类 Exception。

Java 中的异常类层次结构如下：

```
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                java.lang.ArithmeticException
                java.lang.ArrayIndexOutofBoundsException
                java.lang.NullPointerException…
```

此外，还有 EOFException、FileNotFoundException、MalformedURLException 等。

2) 捕获异常

异常处理的典型用法如下，将可能出现异常的代码放在 try 块中，其后由一个或多个 catch 捕获相应异常进行处理，注意只执行第一个匹配的 catch 块，忽略后面的。

```
try {
    //可能出现异常的程序块
}catch(Exception e){
}catch(ArithemticException e){
    //一定不会被执行，因为 ArithemticException 是 Exception 的子类
}catch(EOFException e){
    //也不可能被执行，因为也是 Exception 的子类
}finally{
    //无论是否发生异常或发生什么异常都会被执行
}
```

如果某个方法中所产生的异常该方法自己没有处理，那么可以在调用该方法的方法中进行处理，如果自己处理了，那么调用它的方法就无法得到该异常。

3) 抛出异常

异常除了运行中系统产生的之外，也可以主动抛出异常，用关键字 throw，如 throw new Exception()。注意：throw 只能抛出 Throwable 子类的异常。

4) 带 throws 关键字的方法

带 throws 关键字的方法声明如下：

```
void function() throws Exception{
}
```

强制调用该方法的方法必须处理可能发生的异常，或者将异常重新定向。假定方法 A 带 throws 关键字，而方法 B 中调用了方法 A，则方法 B 中必须有处理方法 A 中可能产生的异常的语句，或者方法 B 也带 throws 关键字，指明调用方法 B 的方法必须处理异常。

10. final 关键字

用 final 关键字定义的常量，在其初始化或第一次赋值后，其值不能被改变。常量必须先有值，然后才能使用。对于常量的第一次赋值只能在构造函数中进行。

final 对象的值不能被改变，指的是该对象不能再指向其他对象，而不是指不能改变当

前对象内部的属性值。

函数参数声明为 final 后，函数中不能改变其值。

final 方法是不能被重置的方法。

final 类不能被继承，其所有方法都是 final 的，但属性可以不是 final 的。

6.2.2 典型例题分析

例 1 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2013 年 5 月试题六)

【说明】(共 15 分)

现要求实现一个能够自动生成求职简历的程序，简历的基本内容包括求职者的姓名、性别、年龄及工作经历。希望每份简历中的工作经历有所不同，并尽量减少程序中的重复代码。

现采用原型模式(Prototype)来实现上述要求，得到如图 6-26 所示的类图。

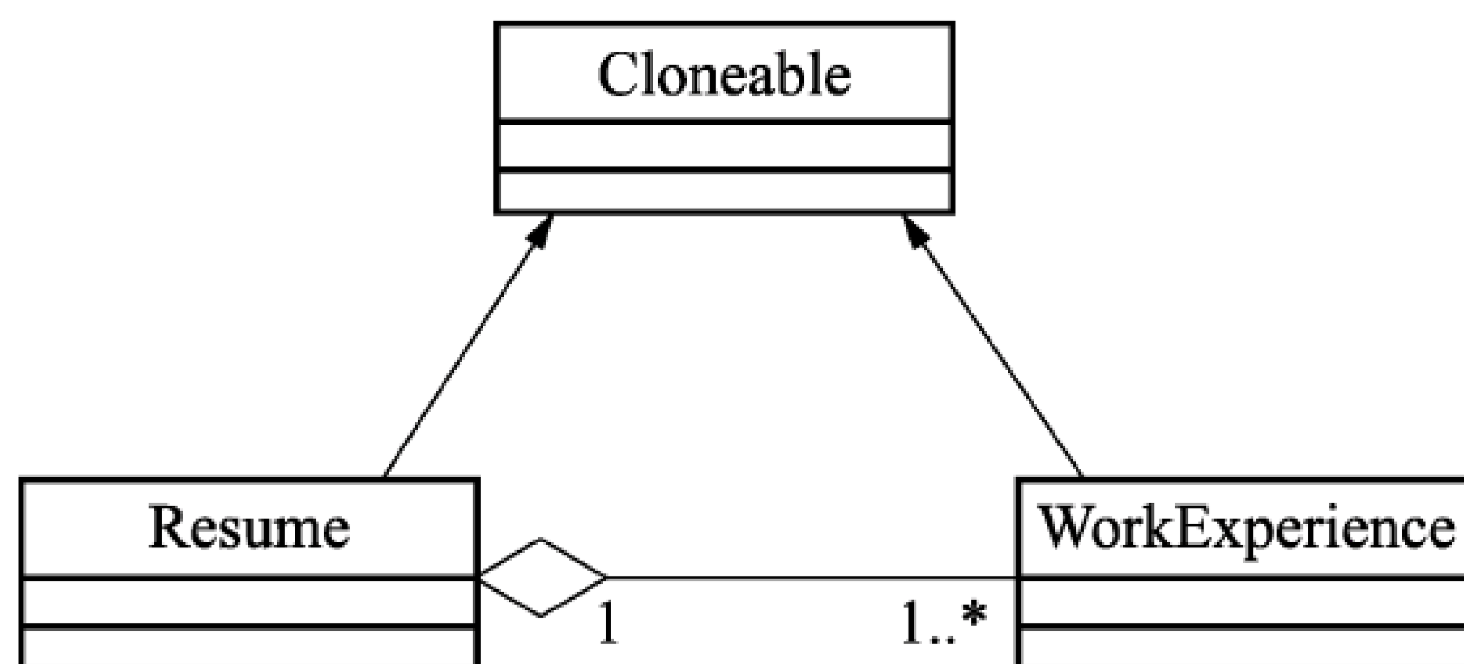


图 6-26 例 1 类图

【Java 代码】

```
Class WorkExperience  (1)  Cloneable{  //工作简历
        Private String workDate;
        Private String company;
        Public  Object  Clone() {
            (2) ;
            obj.workDate=this.workDate;
            Obj.company-this.company;
             Return obj;
         }
}
Class Resume   (3)   Cloneable{   //简历
       Private String name;
       Private String sex;
       Private String age;
       Private WorkExperience work;
       Public Resume(String name){
         This.name=name;      work=new WorkExperience();
       }
       Private Resume(WorkExperience work){
          This.woek= (4) ;
```

```
        }
        Public void SetPersonalInfo(String sex,String age){   /*代码省略*/ }
        Public void SetWorkExperience(String workDate,String company)
        { /*代码省略*/   }
        Public Object Clone(){
              Resume obj=  (5) ;
              //其余代码省略
              Return  obj;
        }
    }
    Class  WorkResume{
       Public static void main(String[]  args){
          Resume  a=new  Resume("张三");
          a.SetPersonalInfo("男","29");
          a.SetWorkExperience("1998~2000","XXX 公司");
          Resume b=  (6) ;
          b.SetWorkExperience("2001~2006","YYY 公司");
       }
    }
```

解析：

本题考查原型模式的概念及应用。原型模式是一种对象创建模型，用原型实体指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另一个可定制的对象，无须知道任何创建的细节。

所有的Java类都继承自java.lang.Object，而object类提供一个Clone()方法，可以将一个Java对象复制一份，因此在Java中可以直接使用Object提供的Clone()方法来实现对象的克隆。能够实现克隆的Java类必须实现一个标识接口Cloneable，表示这个Java类支持复制。

题中WorkExperience类和Resume类需要实现Cloneable接口，故空(1)处和空(3)处应填入implements。WorkExperience中需要实现Clone方法，并将自身复制一份，由下面的代码可知空(2)处应填入WorkExperience obj=new WorkExperience()。Resume类中的私有构造方法实现WorkExperience的深复制，故空(4)处应填入(WorkExperience)work.Clone()。而Resume类中的Clone方法实现自身的复制，故空(5)处应填入new Resume(this.work)。main中实现Resume b对a的复制，故空(6)处应填入a.Clone()。

答案：

(1) implements。　　(2) WorkExperience obj=new WorkExperience()。

(3) implements。　　(4) (WorkExperience)work.Clone()。

(5) new Resume(this.work)。　　(6) a.Clone()。

例 2　阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2012 年 11 月试题六)

【说明】(共 15 分)

现欲开发一个软件系统，要求能够同时支持多种不同的数据库，为此采用抽象工厂模式设计该系统。以 SQL Server 和 Access 两种数据库以及系统中的数据库表 Department 为例，其类图如图 6-27 所示。

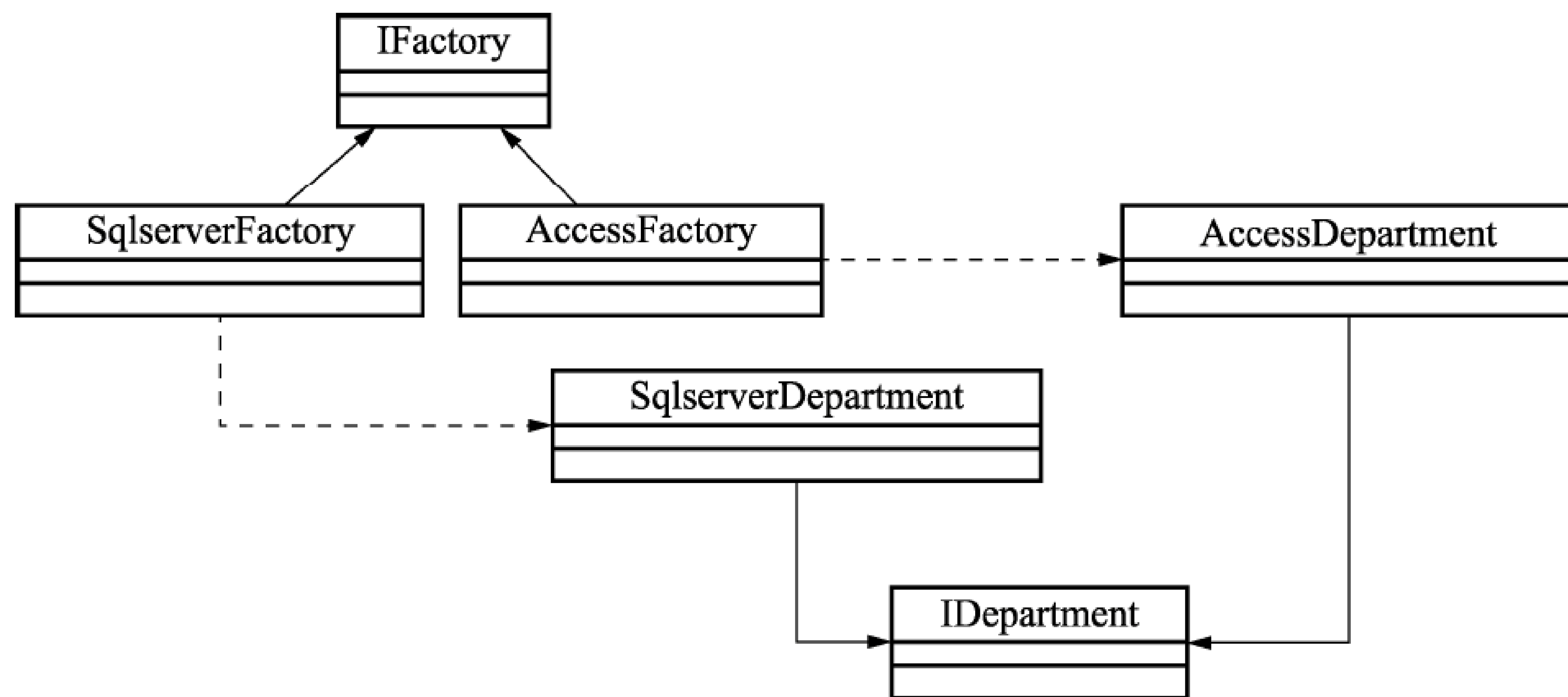


图 6-27　例 2 类图

【Java 代码】

```
import java.util.*;
class Department{/*代码省略*/}
interface IDepartment{
    ___(1)___;
    ___(2)___;
}
class SqlserverDepartment:___(3)___{
public:
    void Insert(Department department){
        System.out.println("Insert a record into Department in SQL Server!\n");
        //其余代码省略
    }
    public Department GetDepartment(int id){
        /*代码省略*/
    }
}
class AccessDepartment:___(4)___{
    public void Insert(Department department){
        System.out.println("Insert a record into Department in ACCESS!\n");
        //其余代码省略
    }
    public Department GetDepartment(int id){
        /*代码省略*/
    }
};
___(5)___{
    ___(6)___;
}
class SqlServerFactory implements IFactory{
    public IDepartment CreateDepartment(){
        return new SqlserverDepartment();
    }
    //其余代码省略
};
```

```
class AccessFactory implements IFactory{
    public IDepartment CreateDepartment(){
        return new AccessDepartment();
    }
    //其余代码省略
};
```

解析：

本题考查抽象工厂设计模式的概念及其应用。

抽象工厂设计模式的意图是：提供一个创建一系列相关或相互依赖的对象，而无须指出它们具体的类。在以下情况应当考虑使用抽象工厂模式。

- 当一个系统要独立于它的产品的创建、组合和表示时。
- 当一个系统要由多个产品系列中的一个来配置时。
- 当需强调一系列相关的产品对象的设计以便进行联合使用时。
- 当想提供一组对象而不显示它们的实现过程，只显示它们的接口时。

抽象工厂设计模式的类图如图 6-28 所示，其中：

- Abstractory 为抽象工厂，声明抽象产品的方法。
- ConcreteFactory 为具体工厂，执行生成抽象产品的方法，生成一个具体的产品。
- Product A 和 Product B 为抽象产品，为一种产品声明接口。
- ProductA1、ProductA2、ProductB1、ProductB2 为具体产品，定义具体工厂生成的具体产品的对象，实现产品接口。

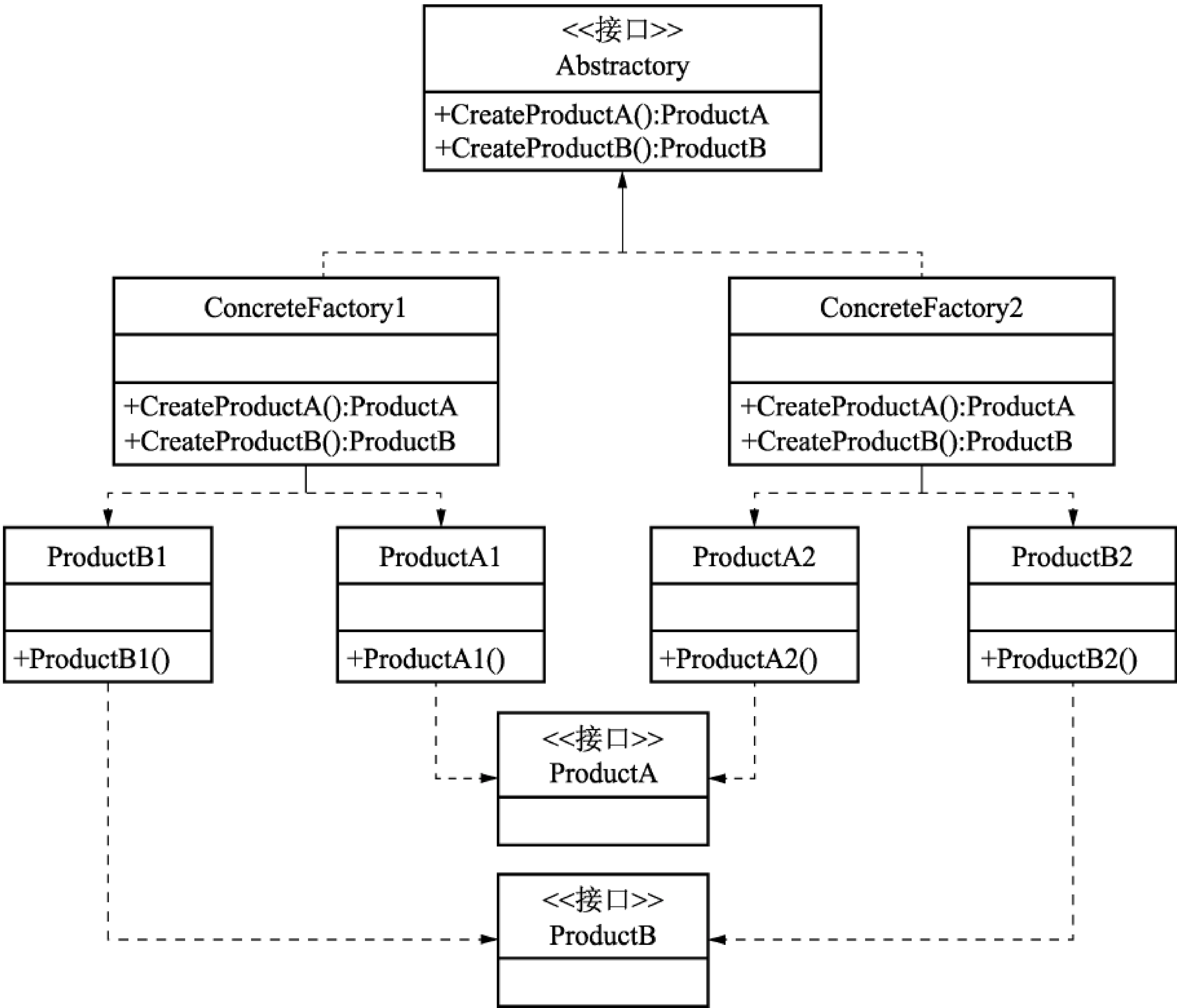


图 6-28　抽象工厂设计模式类图

图 6-27 中的 IFactory 对应图 6-28 中的 Abstractory，SqlserverFactory 和 AccessFactory 对应图 6-28 中的 ConcreteFactory，SqlserverDepartment 和 AccessDepartment 对应图 6-28 中的 ProductA1、ProductA2、ProductB1、ProductB2，而 IDepartment 对应图 6-28 中的 Product A 和 Product B。

空(1)处和空(2)处考查接口 IDepartment 中方法的定义。由其子类 SqlserverDepartment 和 AccessDepartment 中方法的定义，可知空(1)处应填入 void Insert(Department department)，空(2)处应填入 Department GetDepartment(int id)。

空(3)处和空(4)处考查接口 IDepartment 的实现。接口的实现是在一个类的声明中使用关键字“implements”来表示该类使用某个已经定义的接口，然后即可在该类体中使用接口中定义的常量，而且必须实现接口中定义的所有方法。因此，空(3)处和空(4)处都应填入 implements IDepartment。

由于所给程序中缺少接口 IFactory 的定义，因此空(5)处应定义接口 IFactory，应填入 interface IFactory。由于类 SqlserverFactory 和 AccessFactory 中必须实现接口 IFactory 中定义的所有方法，观察这两个类中的方法可知，空(6)处应填入 IDepartment CreateDepartment()。

答案：

(1) void Insert(Department department)。

(2) Department GetDepartment(int id)。

(3) implements IDepartment。

(4) implements IDepartment。

(5) interface IFactory。

(6) IDepartment CreateDepartment()。

例 3 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2012 年 5 月试题六)

【说明】(共 15 分)

某咖啡店卖咖啡时，可以根据顾客的要求在其中加入各种配料，咖啡店会根据所加入的配料来计算费用。咖啡店所供应的咖啡及配料的种类和价格如表 6-6 所示。

表 6-6 咖啡店所供应的咖啡及配料的种类和价格

咖　啡	价格/(元/杯)	配　料	价格/(元/杯)
蒸馏咖啡(Espresso)	25	摩卡(Mocha)	10
深度烘焙咖啡(DarkRoast)	20	奶泡(Whip)	8

现采用装饰器(Decorator)模式来实现计算费用的功能，得到如图 6-29 所示的类图。

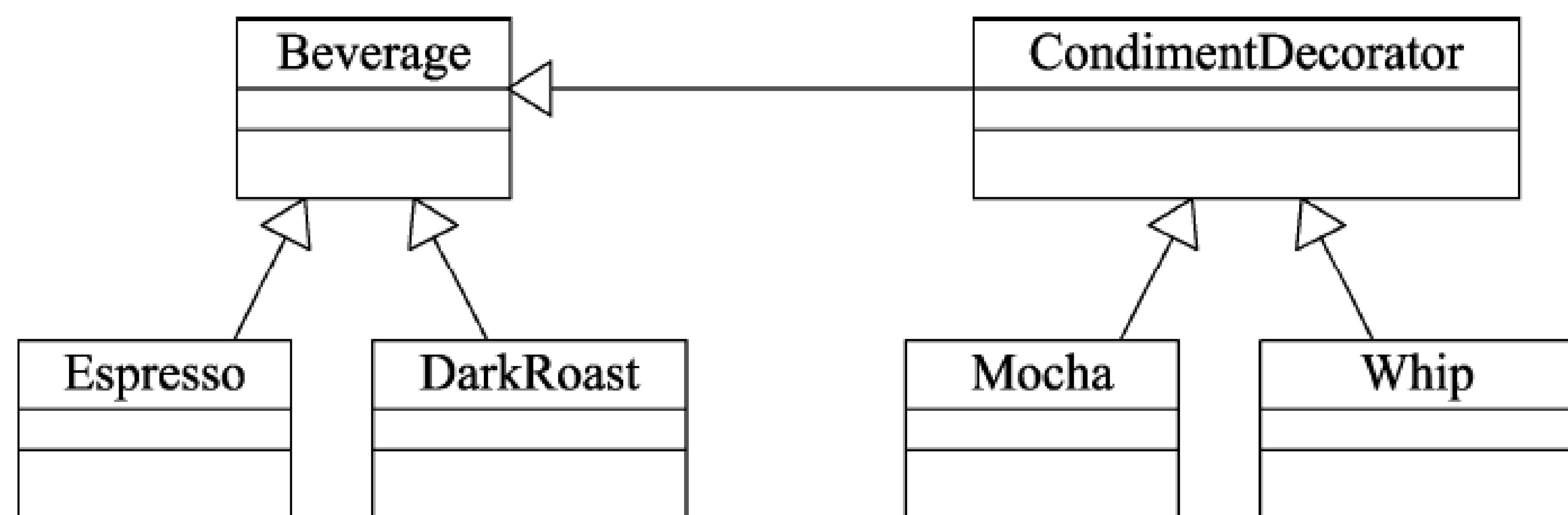


图 6-29 例 3 类图

【Java 代码】

```
import java.util.*;
 (1)  class Beverage { //饮料
    String description = "Unknown Beverage";
    public  (2)  (){return description;}
    public  (3) ;
}
abstract class CondimentDecorator extends Beverage { //配料
   (4) ;
}
class Espresso extends Beverage { //蒸馏咖啡
  private final int ESPRESSO_PRICE = 25;
  public Espresso() { description="Espresso"; }
  public int cost() { return ESPRESSO_PRICE; }
}
class DarkRoast extends Beverage { //深度烘焙咖啡
  private final int DARKROAST_PRICE = 20;
  public DarkRoast() { description = "DarkRoast"; }
  public int cost(){ return DARKROAST PRICE; }
}
class Mocha extends CondimentDecorator { //摩卡
  private final int MOCHA_PRICE = 10;
  public Mocha(Beverage beverage) {
    this.beverage = beverage;
  }
  public String getDescription() {
    return beverage.getDescription() + ", Mocha";
  }
  public int cost() {
    return MOCHA_PRICE + beverage.cost();
  }
}
class Whip extends CondimentDecorator { //奶泡
  private final int WHIP_PRICE = 8;
  public Whip(Beverage beverage) { this.beverage = beverage; }
  public String getDescription() {
    return beverage.getDescription()+", Whip";
  }
  public int cost() { return WHIP_PRICE + beverage.cost(); }
}
public class Coffee {
  public static void main(String args[]) {
    Beverage beverage = new DarkRoast();
    beverage=new Mocha( (5) );
```

```
    beverage=new Whip( (6) );
    System.out.println(beverage.getDescription() +"¥" +beverage.cost());
  }
}
```

编译运行上述程序，其输出结果为：

```
DarkRoast, Mocha, Whip ¥38
```

解析：

装饰器模式描述了如何动态地为一个对象添加职责。该模式采用递归方式组合对象，从而允许添加任意多的对象职责。在下列情况下可以使用装饰器模式。

(1) 在不影响其他对象的情况下，以动态、透明的方式给单个对象添加职责。

(2) 处理那些可以撤销的职责。

(3) 当不能采用生成子类的方法进行扩充时。一种情况是，可能有大量独立的扩展，为支持每一种组合将产生大量的子类，使得子类数目呈爆炸性增长。另一种情况可能是因为类定义被隐藏，或类定义不能用于生成子类。

装饰器模式的结构如图 6-30 所示。

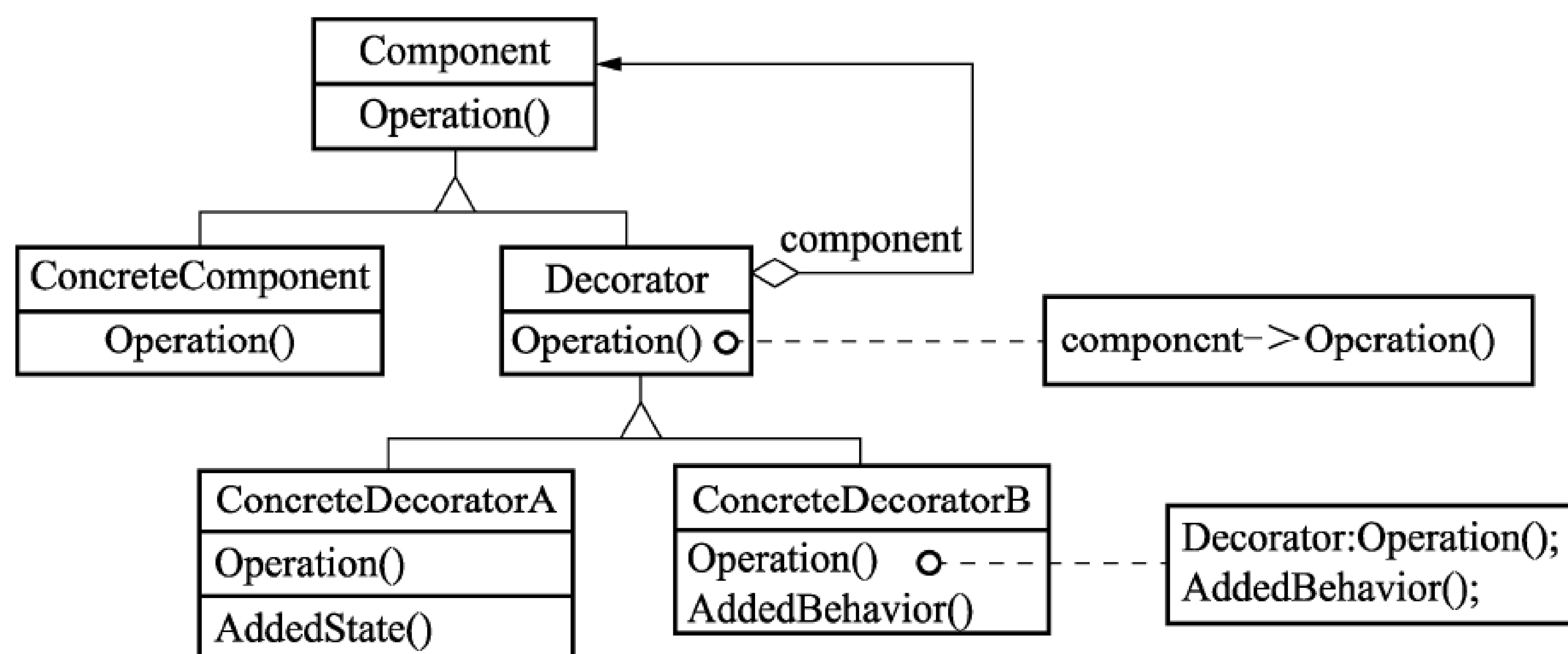


图 6-30　装饰器模式的结构

- 抽象组件角色(Component)：定义一个对象接口，以规范准备接受附加责任的对象，即可以给这些对象动态地添加职责。
- 具体组件角色(ConcreteComponent)：被装饰者，定义一个将要被装饰增加功能的类。可以给这个类的对象添加一些职责。
- 抽象装饰器(Decorator)：维持一个指向构件 Component 对象的实例，并定义一个与抽象组件角色(Component)接口一致的接口。
- 具体装饰器角色(ConcreteDecorator)：向组件添加职责。

图 6-29 中的 Beverage 对应的就是图 6-30 中的抽象类 Component，Espresso 和 DarkRoast 对应的是 ConcreteComponent，CondimentDecorator 对应的是 Decorator，Mocha 和 Whip 扮演的是类 CondimentDecorator 的具体装饰器角色 ConcreteDecorator。

由于类 Beverage 为其子类提供了统一的操作接口，所以将其定义为抽象类。可以通过在类名前加 abstract 关键字来定义抽象类，因此空(1)处应填入 abstract。

空(2)处和空(3)处考查构造函数的定义。从空(2)处构造函数体中返回值的类型及后续的

子类继承程序可知，空(2)处应填入 String getDescription；从 public int cost() { return ESPRESSO_PRICE;}可以看出，cost()函数的返回值为常量，因此(3)处应填入 abstract int cost()。

空(4)处考查对 CondimentDecorator 的定义，在该类中声明一类成员变量，并在 this.beverage = beverage 和 return beverage.getDescription() + ", Mocha"中加以使用，因此空(4)处应填入 Beverage beverage。

空(5)处和空(6)处考查实例化类模板的方法。类模板必须在实例化后才能使用。实例化类模板时，要给出类型实参。从类图可知，空(5)处和空(6)处均应填入 beverage。

答案：

(1) abstract。　(2) String getDescription。　(3) abstract int cost()。

(4) Beverage beverage。　(5) beverage。　(6) beverage。

例 4　阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2011 年 11 月试题六)

【说明】(共 15 分)

某大型商场内安装了多个简易的纸巾售卖机，自动售出 2 元钱一包的纸巾，且每次仅售出一包纸巾。纸巾售卖机的状态图如图 6-31 所示。

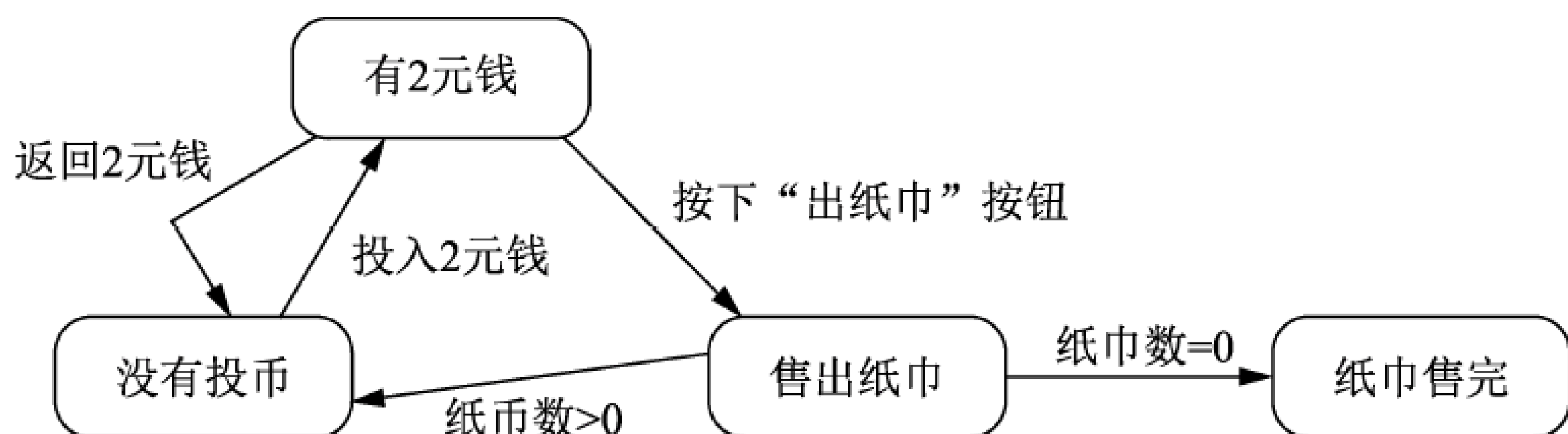


图 6-31　纸巾售卖机状态图

采用状态(State)模式来实现该纸巾售卖机，得到如图 6-32 所示的类图。其中类 State 为抽象类，定义了投币、退币、出纸巾等方法接口。类 SoldState、SoldOutState、NoQuarterState 和 HasQuarterState 分别对应图 6-31 中纸巾售卖机的 4 种状态：售出纸巾、纸巾售完、没有投币、有 2 元钱。

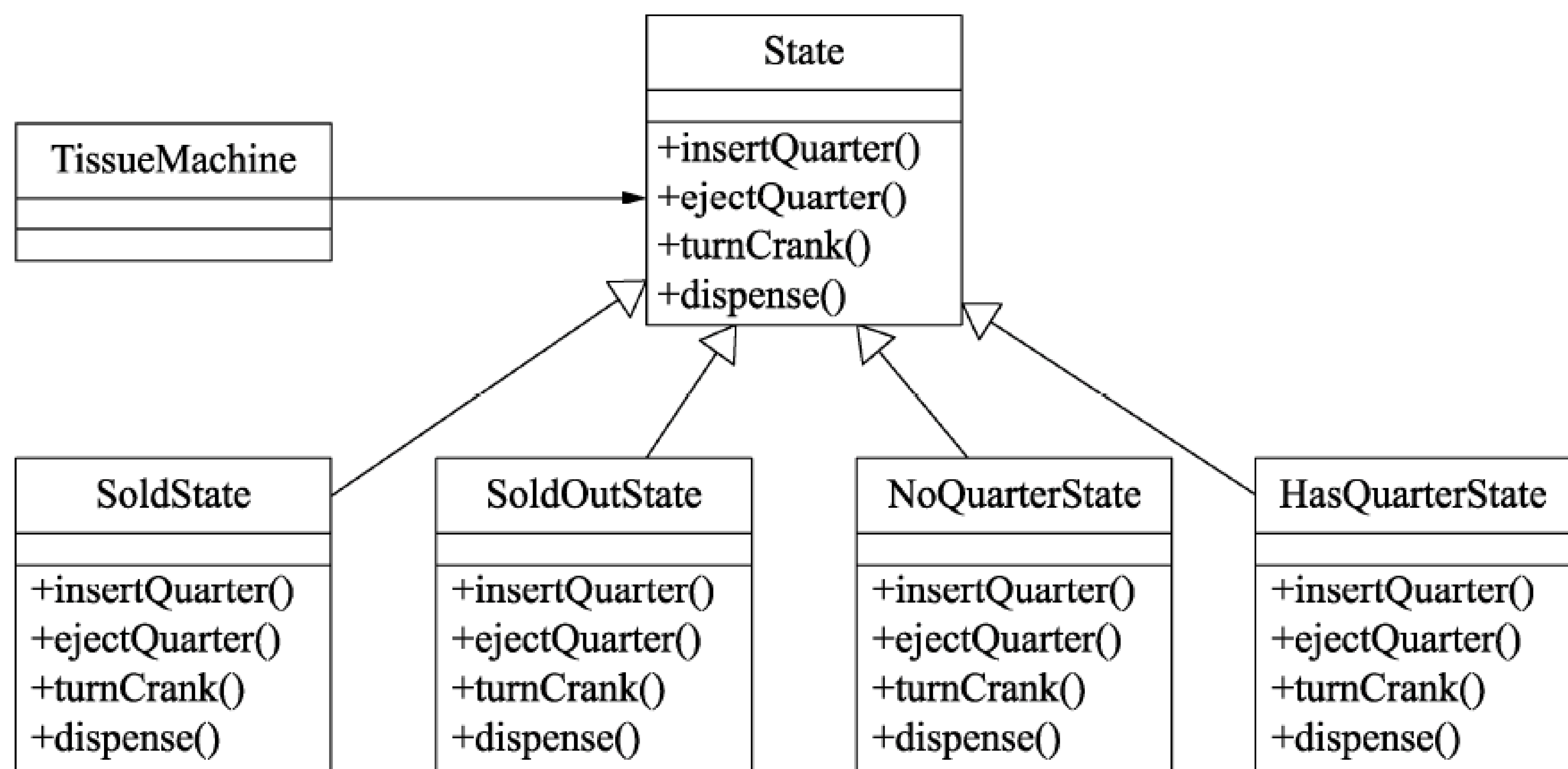
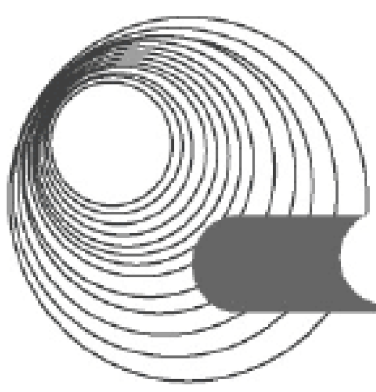


图 6-32　例 4 类图



【Java 代码】

```
import java.util.*
interface State{
    public void insertQuarter();    //投币
    public void ejectQuarter();     //退币
    public void turnCrank();        //按下“出纸巾”按钮
    public void dispense();         //出纸巾
}
class TissueMachine{
    (1)  soldOutState,noQuarterState,hasQuarterState,soldState,state;
    state=soldOutState;
    int count=0;                        //纸巾数
    public TissueMachine(int number) {/*实现代码省略*/}
    public State getHasQuarterState() {return hasQuarterState;}
    public State getNoQuarterState() {return noQuarterState;}
    public State getSoldState() {return soldState;}
    public State getSoldOutState() {return soldOutState;}
    int getCount {return count;}
    //其余代码省略
};
class NoQuarterState implement State {
   TissueMachine tissueMachine;
   public void insertQuarter(){
       tissueMachine.setState((2));
   }
//构造方法以及其余代码省略
}
class HasQuarterState implement State {
   TissueMachine tissueMachine;
   public void ejectQuarter(){
       tissueMachine.setState((3));
   }
//构造方法以及其余代码省略
}
class SoldState implement State {
   TissueMachine tissueMachine;
   public void dispense(){
       if(tissueMachine.getCount()>0){
        tissueMachine.setState((4));
       }
    else{
        tissueMachine.setState((5));
       }
   }
}
```

解析：

空(1)处：根据题意，本题使用的是状态模式，判断纸巾售卖机的状态，根据不同的状

态执行不同的动作。State 定义了纸巾售卖机所对应的一些状态，如售出纸巾、纸巾售完等。类 SoldOutState、NoQuarterState、HasQuarterState、SoldState 均由类 State 派生而来。

空(2)处：public void insertQuarter()定义了一个“投币”的方法：在“没有投币”状态下，客户投币的方法。tissueMachine.setState 是改变纸巾售卖机的状态，此时，客户已投入 2 元钱，故将此时的状态改为“有 2 元钱”的状态，纸巾售卖机调用“有 2 元钱”状态的方法即可。

空(3)处：public void ejectQuarter()定义了一个“退币”的方法：在“有 2 元钱”状态下，用户按下退币按钮，纸巾售卖机将此时的状态改为“没有投币”状态，故直接调用 NoQuarterState()即可。

空(4)处：根据纸巾售卖机状态图可知，当售出纸巾并且纸巾数量仍大于 0 时，将返回“没有投币”状态，同空(3)处的分析，此处应调用 NoQuarterState()。

空(5)处：根据纸巾售卖机状态图可知，当售出纸巾并且纸巾数量等于 0 时，将返回“纸巾售完”状态，此处用了 if…else…判断语句，当纸巾数量仍大于 0 时，返回“没有投币”状态，否则，纸巾数量一定等于 0。空(5)处是 else 下的一条语句，故此处是判断 count 为 0 时纸巾售卖机的状态，显然，应调用 SoldOutState()。

答案：

(1) State。

(2) tissueMachine.getHasQuarterState()或 new HasQuarterState()或 tissueMachine.HasQuarterState()。

(3) tissueMachine.getNoQuarterState()或 new NoQuarterState()或 tissueMachine.NoQuarterState()。

(4) tissueMachine.getNoQuarterState()或 new NoQuarterState()或 tissueMachine.NoQuarterState()。

(5) tissueMachine.getSoldOutState()或 new SoldOutState()或 tissueMachine.SoldOutState()。

例 5　阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2011 年 5 月试题六)

【说明】(共 15 分)

某饭店在不同的时段提供多种不同的餐饮，其菜单结构图如图 6-33 所示。

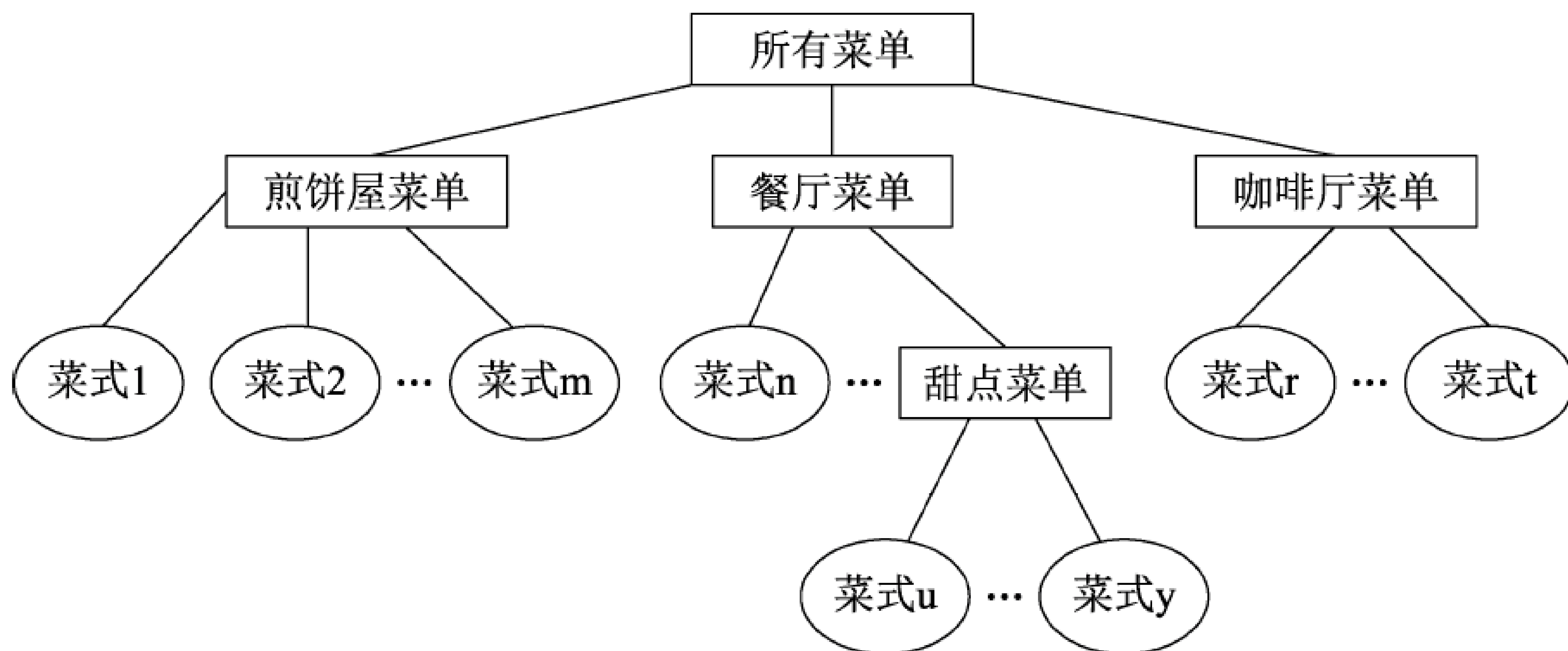


图 6-33　菜单结构图

现在采用组合(Composition)模式来构造该饭店的菜单，使得饭店可以方便地在其中添加新的餐饮形式，得到如图 6-34 所示的类图。其中 MenuComponent 为抽象类，定义了添加(add)新菜单和打印饭店所有菜单信息(print)的方法接口。类 Menu 表示饭店提供的每种餐饮形式

的菜单，如煎饼屋菜单、咖啡屋菜单等。每种菜单中都可以添加子菜单，例如图 6-33 中的甜点菜单。类 MenuItem 表示菜单中的菜式。

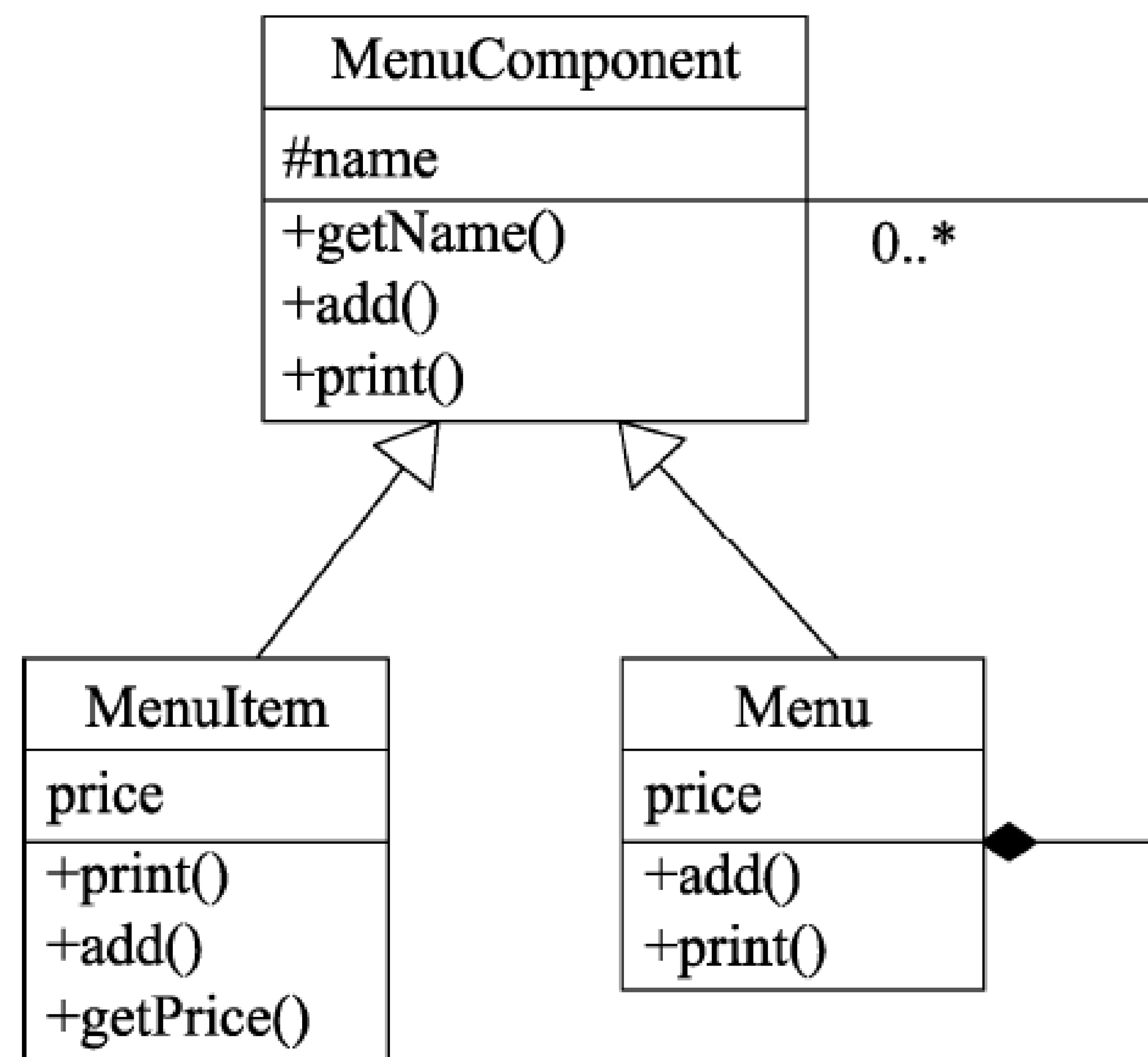


图 6-34　例 5 类图

【Java 代码】

```
import java.util.*
(1) MenuComponent{
    protected String name;
 (2) ;//添加新菜单
    public abstract void print();      //打印菜单信息
    public String getName(){return name;}
}
class MenuItem extends MenuComponent{
    private double price;
    public MenuItem(String name,double price){
    this.name=name; this.price=price;
}
public double getPrice(){return price;}
public void add(MenuComponent menuComponent){return;}//添加新菜单
public void print(){
    System.out.print(" "+getName());
    System.out.print1(","+getPrice());}
    class Menu extends MenuComponent{
        private List<MenuComponent>menuComponents =new ArrayList<Menu
Component>();
        public Menu(String name){this.name=name;}
public void add(MenuComponent menuComponent){
menuComponents. (3) ;
}
public void print(){
System.out.print("\n"+getName());
System.out.println(","+"---------------------");
Iterator iterator=menuComponents.iterator();
while(iterator.hasNext()){
```

```
MenuComponent menuComponent=(MenuComponent)iterator.next();
 (4) ;
}
}
}
class MenuTestDrive{
public static void main(String args[]){
MenuComponent allMenus=new Menu("ALL MENUS");
MenuComponent dinerMenu=new Menu("DINER MENU");
…//创建更多的 Menu 对象，此处代码省略
allMenus.add(dinerMenu);  //将 dinerMenu 添加到餐厅菜单中
…//为餐厅增加更多的菜单，此处代码省略
 (5) ;//打印饭店所有菜单的信息
}
}
```

解析：

组合模式将对象组合成树型结构以表示“整体–部分”的层次结构，其中的组合对象使得用户可以组合基元对象以及其他的组合对象，从而形成任意复杂的结构。组合模式使得用户对单个对象和组合对象的使用具有一致性。

组合模式的结构如图 6-35 所示。

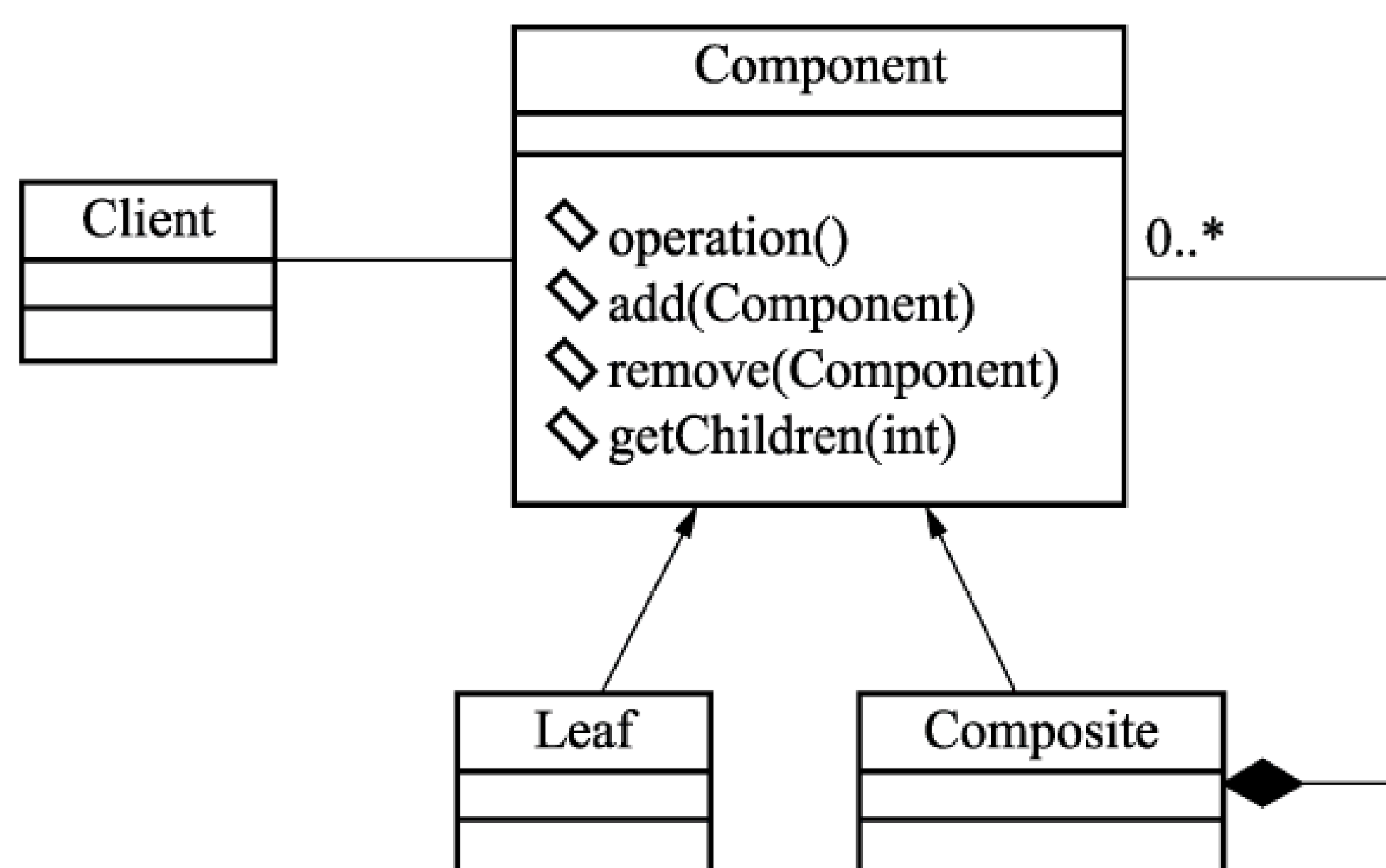


图 6-35　组合模式的结构

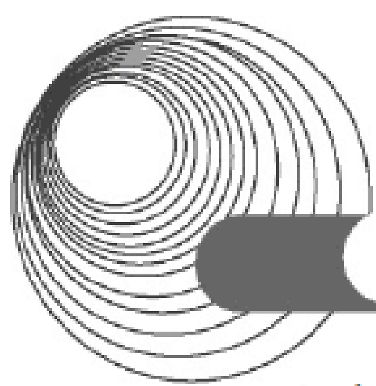
图中各部分说明如下。

- 类 Component 为组合中的对象声明接口，在适当的情况下，实现所有类公有接口的缺省行为，声明一个接口用于访问和管理 Component 的子部件。
- 类 Leaf 在组合中表示叶节点对象，叶节点没有子节点，并在组合中定义图元对象的行为。
- 类 Composite 定义有子部件的那些部件的行为，存储子部件，并在 Component 接口中实现与子部件有关的操作。
- 类 Client 通过 Component 接口操纵组合部件的对象。

下列情况可以使用组合模式。

(1) 表示对象的整体—部分层次结构。

(2) 希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有



对象。

本试题将组合模式应用到饭店菜单的构造中。图 6-34 中的类 MenuComponent 对应图 6-35 中的 Component，MenuItem 对应 Leaf，Menu 对应 Composite。在实现时，通常都会把 Component 定义为抽象类。

在 Java 中，用 abstract 关键字限定的类即为抽象类，所以空(1)处应填入 abstract class。根据注释，空(2)处应该定义功能为“添加新菜单”的成员函数。在子类 MenuItem 和 Menu 中都可以看到 add 成员函数，说明子类中重置了父类中的成员函数。所以空(2)处应填入 public abstract void add(MenuComponent menuComponent)。

由图 6-34 可以看出，Menu 中包含了 MenuComponent 的对象集合。程序中用 Java 中的 list 来实现这个聚集关系，这样就可以利用 list 中提供的各种方法了。list 中用于添加元素的方法是 add，所以空(3)处应填入 add(menuComponent)。

空(4)处出现在方法 print 中，其功能是打印出所有菜单的信息。这里使用了 list 中的迭代器类 iterator 遍历每个子菜单，并调用子菜单中定义的 print 方法打印该子菜单的信息。空(4)处应填入 menuComponent.print()。

为了能够在 main 中打印出所有的菜单信息，必须使用表示菜单结构中最顶层菜单的对象来调用 print，因此空(5)处应填入 allMenus.print()。

答案：

(1) abstract class 或 public abstract class。

(2) public abstract void add(MenuComponent menuComponent)。

或 abstract void add(MenuComponent menuComponent)。

或 protected abstract void add(MenuComponent menuComponent)。

(3) add(menuComponent)。

(4) menuComponent.print()。

(5) allMenus.print()。

例 6 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2010 年 11 月试题六)

【说明】(共 15 分)

某公司的组织结构图如图 6-36 所示，现采用组合(Composition)设计模式来设计，得到如图 6-37 所示的类图。

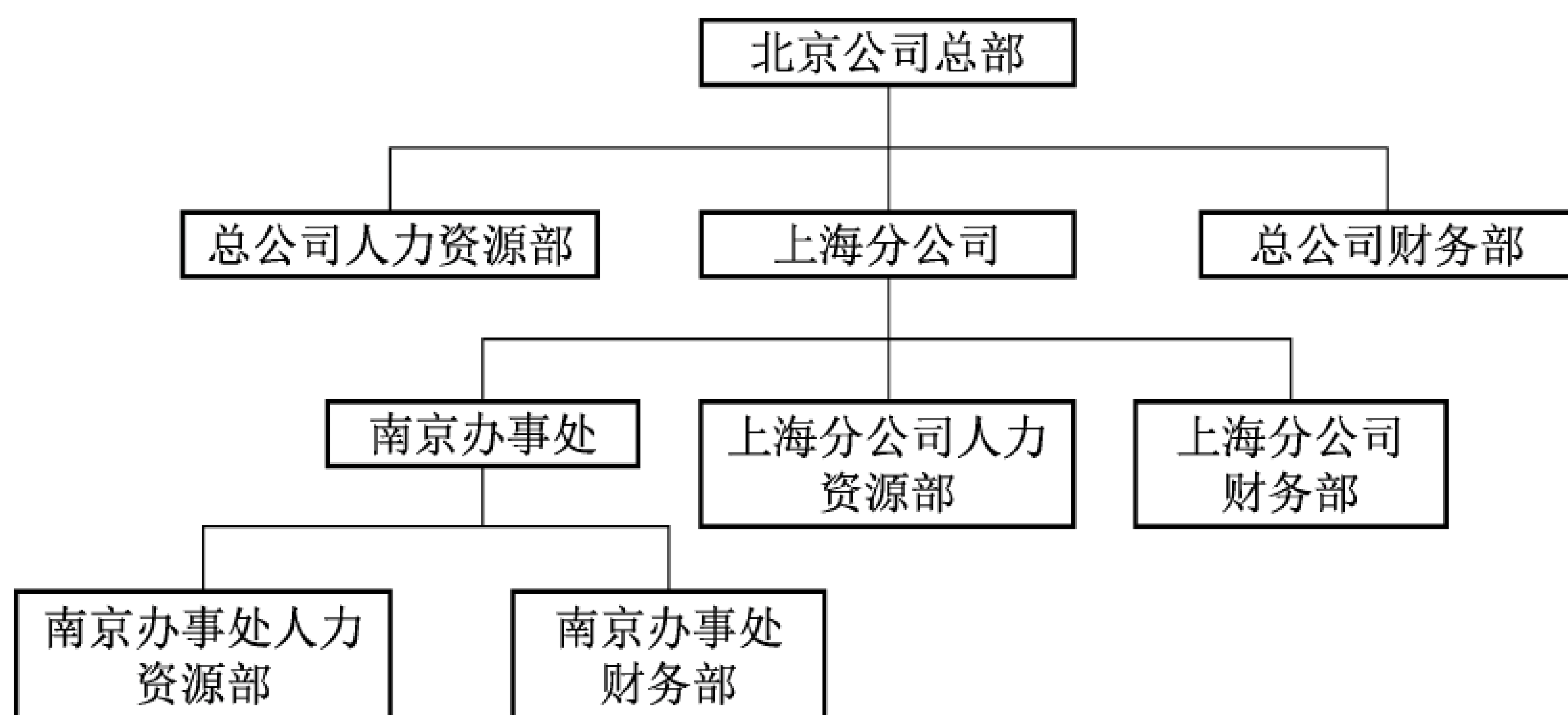


图 6-36 组织结构图

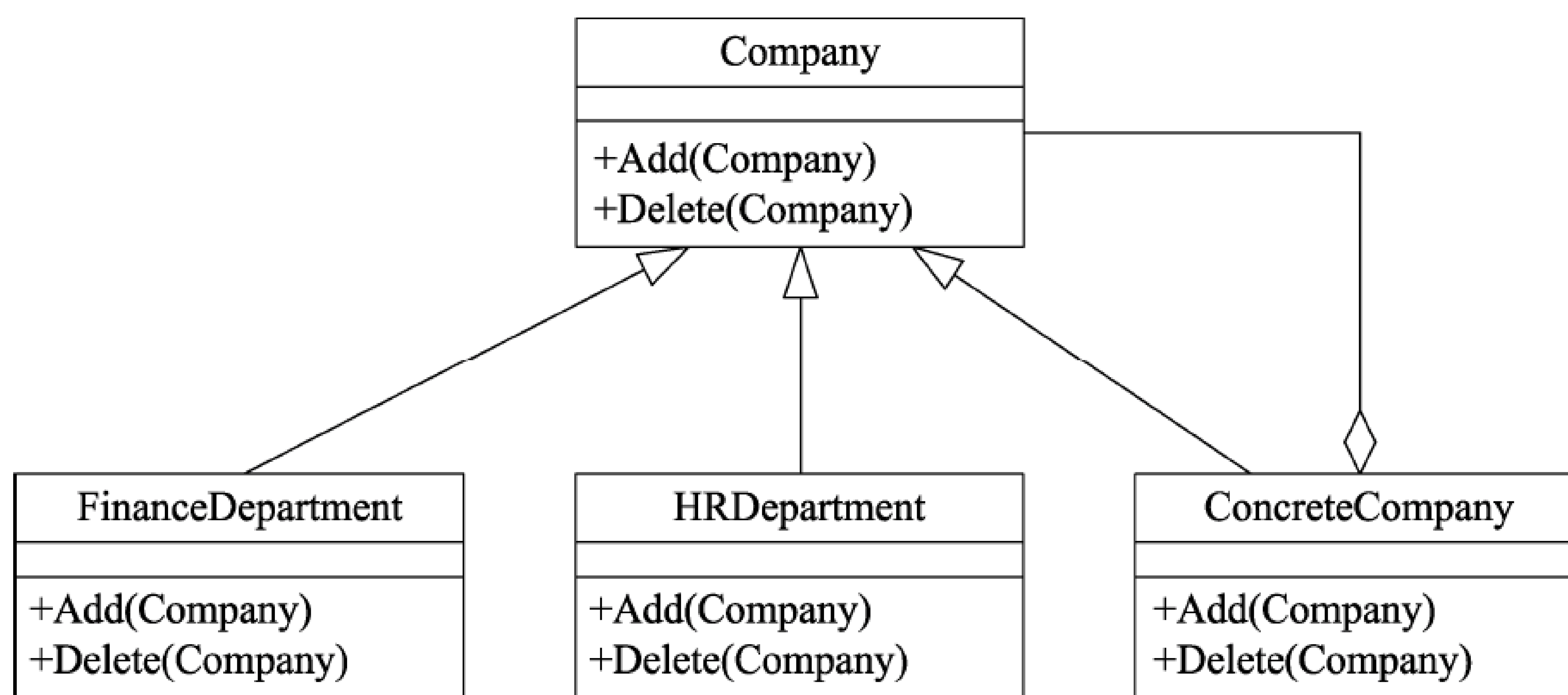


图 6-37　例 6 类图

其中 Company 为抽象类，定义了在组织结构图上添加(Add)和删除(Delete)分公司/办事处或者部门的方法接口。类 ConcreteCompany 表示具体的分公司或者办事处，分公司或办事处下可以设置不同的部门。类 HRDepartment 和 FinanceDepartment 分别表示人力资源部和财务部。

【Java 代码】

```
import java.util.*:
    (1)  Company{
    protected String name;
    public Company(String name){ (2) =name;}
    public abstract void Add(Company c);//增加子公司、办事处或部门
    public abstract void Delete(Company c); //删除子公司、办事处或部门
}
class ConcreteCompany extends Company{
  private List< (3) > children=new ArrayList< (4) >();
                                              //存储子公司、办事处或部门
  public ConcreteCompany(String name){super(name);}
  public void Add(Company c){ (5) .add(c);}
  public void Delete(Company c){ (6) .remove(c);}
}
class HRDepartment extends Company{
    public HRDepartment(String name){super(name);}
    //其他代码省略
}
class FinanceDepartment extends Company{
    public FinanceDepartment(String name){super(name);}
    //其他代码省略
}
public class Test{
    public static void main(String[] args){
    ConcreteCompany root=new ConcreteCompany("北京总公司");
    root.Add(new HRDepartment("总公司人力资源部"));
    root.Add(new FinanceDepartment("总公司财务部"));
```

```
        ConcreteCompany comp=new ConcreteCompany("上海分公司");
        comp.Add(new HRDepartment("上海分公司人力资源部"));
        comp.Add(new FinanceDepartment("上海分公司财务部"));
        (7);
        ConcreteCompany comp1=new ConcreteCompany("南京办事处");
        comp1.Add(new HRDepartment("南京办事处人力资源部"));
        comp1.Add(new FinanceDepartment("南京办事处财务部");
        (8);//其他代码省略
        }
    }
```

解析：

Company 为抽象类，所以空(1)处肯定为 abstract class。空(2)所在的语句为构造函数，用来对 name 字段进行初始化。子公司、办事处或部门都是 Company 这个抽象类的具体实现，所以空(3)处为 Company。空(4)处为 Company。空(5)处所在的语句的作用是向 Company 列表的实例 children 中添加节点。空(6)处所在的语句的作用从 Company 列表的实例 children 中删除节点。空(7)处所在的语句的作用是把上海分公司这个子节点加入到北京公司总部这个根节点中。空(8)处所在的语句的作用是将南京办事处这个子节点加入到上海分公司这个父节点中。

答案：

(1) abstract class。(2) this.name。(3) Company。(4) Company。

(5) children。(6) children。(7) root.Add(comp)。(8) comp.Add(comp1)。

例 7 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2010 年 5 月试题六)

【说明】(共 15 分)

某软件公司现欲开发一款飞机飞行模拟系统，该系统主要模拟不同种类飞机的飞行特征与起飞特征。需要模拟的飞机种类及其特征如表 6-7 所示。

表 6-7 需要模拟的飞机种类及其特征

飞机种类	起飞特征	飞行特征
直升机(Helicopter)	垂直起飞(VerticalTakeOff)	亚音速飞行(SubSonicFly)
客机(AirPlane)	长距离起飞(LongDistanceTakeOff)	亚音速飞行(SubSonicFly)
歼击机(Fighter)	长距离起飞(LongDistanceTakeOff)	超音速飞行(SuperSonicFly)
鹞式战斗机(Harrier)	垂直起飞(VerticalTakeOff)	超音速飞行(SuperSonicFly)

为支持将来模拟更多种类的飞机，采用策略设计模式(Strategy)设计的类图如图 6-38 所示。

图 6-38 中，AirCraft 为抽象类，描述了抽象的飞机，而类 Helicopter、AirPlane、Fighter 和 Harrier 分别描述具体的飞机种类，方法 fly()和 takeOff()分别表示不同飞机都具有飞行特征和起飞特征；类 FlyBehavior 与 TakeOffBehavior 为抽象类，分别用于表示抽象的飞行行为与起飞行为；类 SubSonicFly 与 SuperSonicFly 分别描述亚音速飞行和超音速飞行的行为；类 VerticalTakeOff 与 LongDistanceTakeOff 分别描述垂直起飞与长距离起飞的行为。

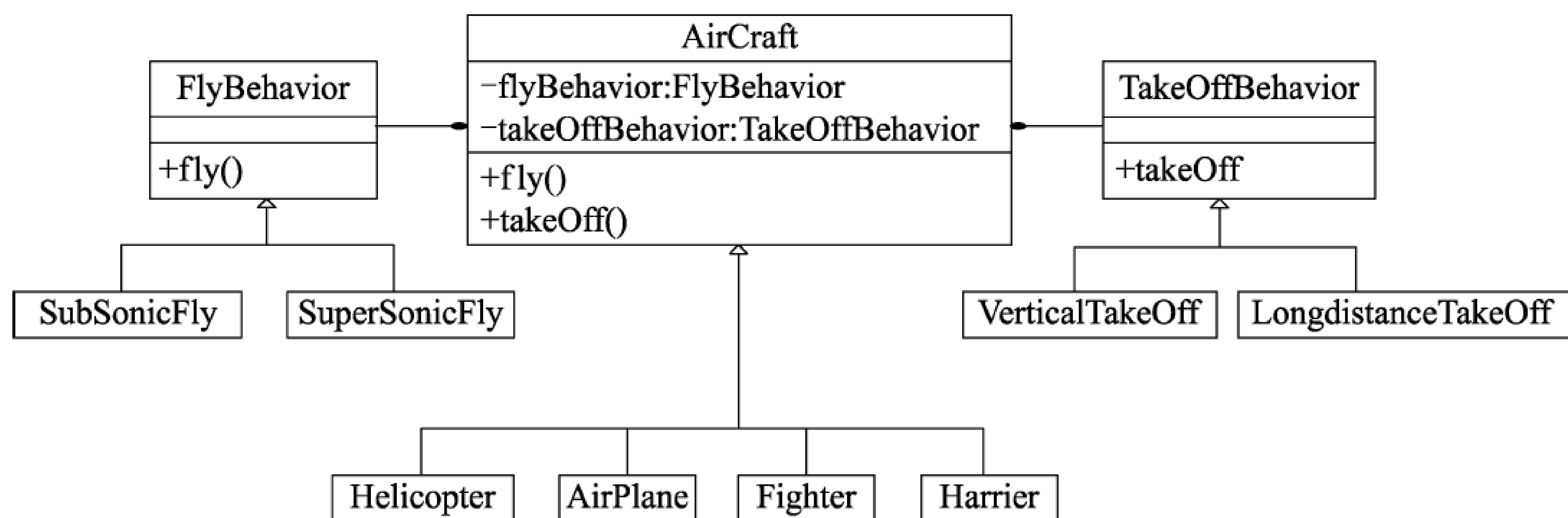


图 6-38　例 7 类图

【Java 代码】

```
interface FlyBehavior {
 public void fly();
};
class SubSonicFly implements FlyBehavior{
 public void fly(){ System.out.println("亚音速飞行! "); }
};
class SuperSonicFly implements FlyBehavior{
 public void fly(){ System.out.println("超音速飞行! " ); }
};
interface TakeOffBehavior {
 public void takeOff();
};
class VerticalTakeOff implements TakeOffBehavior {
 public void takeOff (){ System.out.println("垂直起飞! " ); }
};
class LongDistanceTakeOff implements TakeOffBehavior {
 public void takeOff(){ System.out.println("长距离起飞! "); }
};
abstract class AirCraft {
 protected __(1)__;
 protected __(2)__;
 public void fly(){__(3)__; }
 public void takeOff() {__(4)__; };
};
class Helicopter__(5)__AirCraft{
 public Helicopter(){
 flyBehavior = new __(6)__;
 takeOffBehavior = new __(7)__;
 }
};
//其他代码省略
```

解析：

本题考查设计模式中的策略设计模式。

从本题的叙述中可以看出，存在 4 种不同的飞机类型，但每种飞机类型的起飞特征和飞行特征并不完全相同，这就使得我们很难采用比较直接的方法来实现重用。例如，定义一个抽象的飞机类，实现飞机的起飞特性，然后 4 种飞机直接重用该特征。但是，我们可以观察到，尽管飞机的起飞特征和飞机特征有所不同，但有一点可以肯定的是，每一种飞机都具备了飞行特征和起飞特征。因此，可以抽象出一个飞机类，其中含有飞行特征与起飞特征，但关于两个特征的实现要单独抽取出来，所以又形成了 FlyBehavior 类和 TakeOffBehavior 类，分别表示抽象的飞行和起飞特征，而这两个类的子类则分别实现不同的起飞和飞行特征，最终转化为，在创建一个具体的飞机时，给其配上不同的起飞特征和飞行特征即可。

本题中的空(1)处和空(2)处应该填写成员变量，根据图 6-38 可以得知，此处应该表示的是飞行特征和起飞特征变量。空(3)和空(4)处需要实现飞行特征与起飞特征，但 AirCraft 是抽象的类，所以把实现代理给指针变量。Helicopter 类需要指定由父类继承而来的成员变量的初始值，因为 Helicopter 的特征是垂直起飞和亚音速飞行，因此生成这两个特征的对象，分别赋值给 flyBehavior 和 takeOffBehavior 变量。

答案：

(1) FlyBehavior flyBehavior。

(2) TakeOffBehavior takeOffBehavior。

(3) flyBehavior.fly()。

(4) takeOffBehavior.takeOff()。

(5) extends。

(6) SubSonicFly()。

(7) VerticalTakeOff()。

例 8 阅读以下说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2013 年 11 月试题六)

【说明】

欲开发一个绘图软件，要求使用不同的绘图程序绘制不同的图形。以绘制直线和圆形为例，对应的绘图程序如表 6-8 所示。

表 6-8 不同的绘图程序

	DP1	DP2
绘制直线	draw_a_line(x1,y1,x2,y2)	drawline(x1,x2,yl,y2)
绘制圆	draw_a_circle(x,y,r)	drawcircle(x,y,r)

该绘图软件的扩展性要求，将不断扩充新的图形和新的绘图程序。为了避免出现类爆炸的情况，现采用桥接(Bridge)模式来实现上述要求，得到如图 6-39 所示的类图。

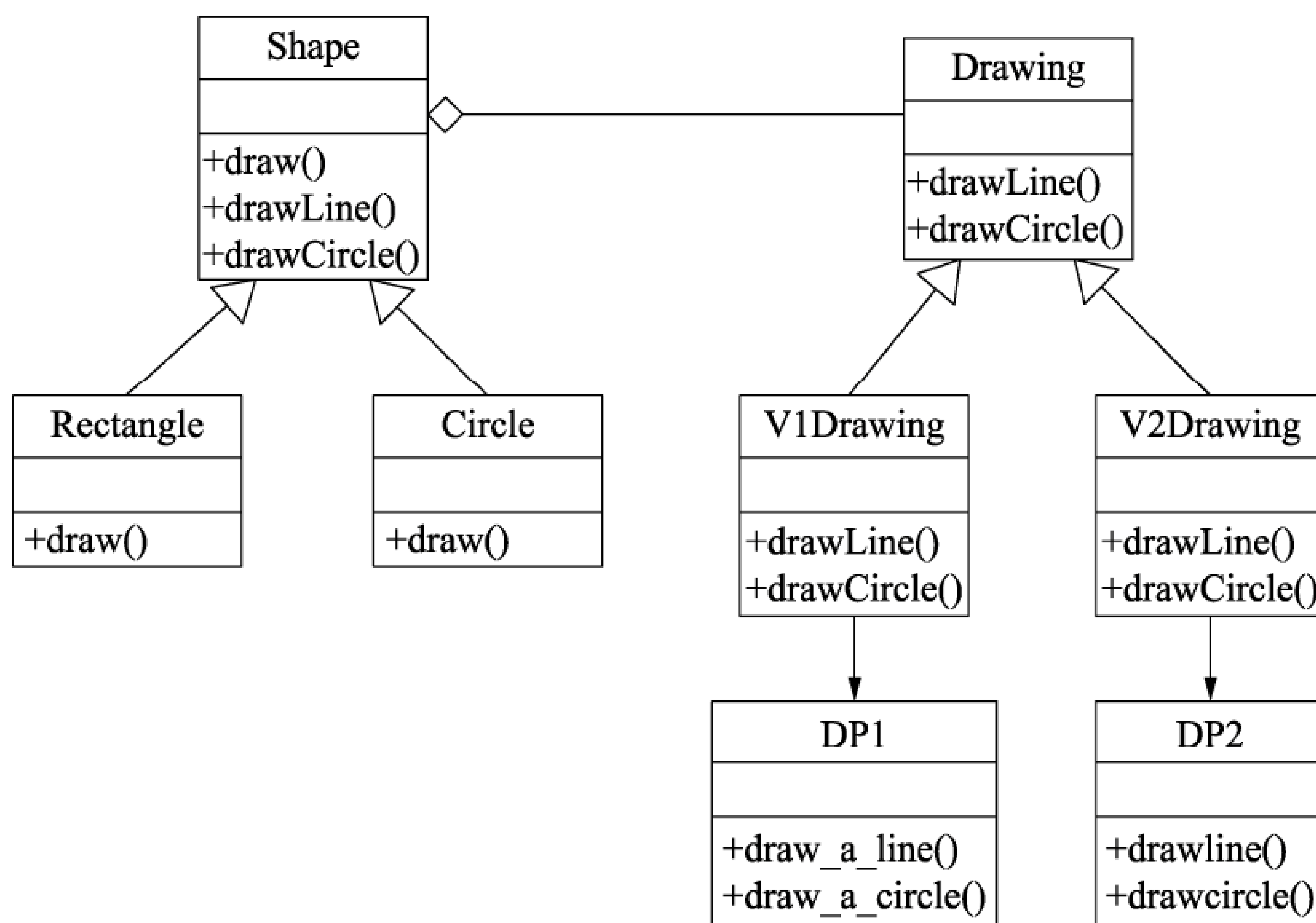


图 6-39　某绘图软件类图

【Java 代码】

```
____(1)____ Drawing{
____(2)____;
____(3)____;
}
class DP1{
    static public void draw_a_line(double x1,double y1,double x2,double y2)
        {/* 代码省略 */}
    static public void draw_a_circle(double x,double y,double r) {/* 代码省略 */}
};
class DP2{
    static public void drawline(double x1,double x2,double y1,double y2)
        {/* 代码省略 */}
    static public void drawcircle(double x,double y,double r) {/* 代码省略 */}
};
class V1Drawing implements Drawing{
    public void drawLine(double x1,double y1,double x2,double y2) {/* 代码省略 */}
    public void drawCircle(double x,double y,double r) {____(4)____;    }
};
class V2Drawing implements Drawing{
    public void drawLine(double x1,double y1,double x2,double y2) {/* 代码省略 */}
    public void drawCircle(double x,double y,double r) {____(5)____;    }
};
abstract class Shape{
    private Drawing _dp;
____(6)____;
    Shape(Drawing dp){_dp=dp;}
    public void drawLine(double x1,double y1,double x2,double y2){
        _dp.drawLine(x1,y1,x2,y2);    }
```

```
    public void drawCircle(double x,double y,double r){_dp.drawCircle(x,y,r);}
};
class Rectangle extends Shape{
    private double _x1,_x2,_y1,_y2;
    public Rectangle(Drawing dp,double x1,double y1,double x2,double y2)
        {/* 代码省略 */}
    public void draw(){  /* 代码省略 */ }
};
class Circle extends Shape{
    private double _x,_y,_r;
    public Circle(Drawing dp,double x,double y,double r) { /* 代码省略 */ }
    public void draw(){  drawCircle(_x,_y,_r);    }
};
```

解析：

本年的试题六和试题五其实是相同的，只是实现方法不同，本题考查 Bridge 桥接模式的概念及应用，将抽象与其实现解耦，使它们都可以独立地变化。大致意思是说：将一组实现与另一组使用它们的对象分离。Java 语言中声明抽象函数用 abstract，且每个函数都需要声明访问类型。对照试题五的代码我们可以看出：空(1)为 interface。这里的实现指的是抽象类及其派生类用来实现自己的对象(而不是抽象类的派生类，这些派生类被称为具体类)。Drawing 是一个虚拟基类，里面包含了希望不同策略实现的算法，派生类 V1Drawing、V2Drawing 都派生自 Drawing，对基类中的希望实现的算法都作了具体实现，且它们都含有 drawLine 和 drawCircle 函数，所以 Draw 类中缺失的应该是这两个算法，于是空(2)为 void drawLine(double x1,double y1,double x2,double y2)，空(3)为 void drawCircle(double x,double y,double r)。DP1 和 DP2 中包含了绘制 Line 和 Circle 的具体实现的 Static 方法，因此在 V1Drawing、V2Drawing 类中可以直接调用它们。空(4)为 DP1.draw_a_circle(x,y,r)。空(5)为 DP2.drawCircle(x,y,r)。Shape 类派生出 Rectangle 类和 Circle 类，里面都含有 draw 方法，但是具体 draw 方法的实现却不相同，所以空(6)为 abstract public void draw()。

答案：

(1) interface。

(2) void drawLine(double x1,double y1,double x2,double y2)。

(3) void drawCircle(double x,double y,double r)。

(4) DP1.draw_a_circle(x,y,r)。

(5) DP2.drawCircle(x,y,r)。

(6) abstract public void draw()。

例 9 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2014 年 11 月试题六)

【说明】

某灯具厂商欲生产一个灯具遥控器，该遥控器具有 7 个可编程的插槽，每个插槽都有开关按钮，对应着一个不同的灯。利用该遥控器能够统一控制房间中该厂商所有品牌灯具的开关，现采用 Command(命令)模式实现该遥控器的软件部分。Command 模式的类图如图 6-40 所示。

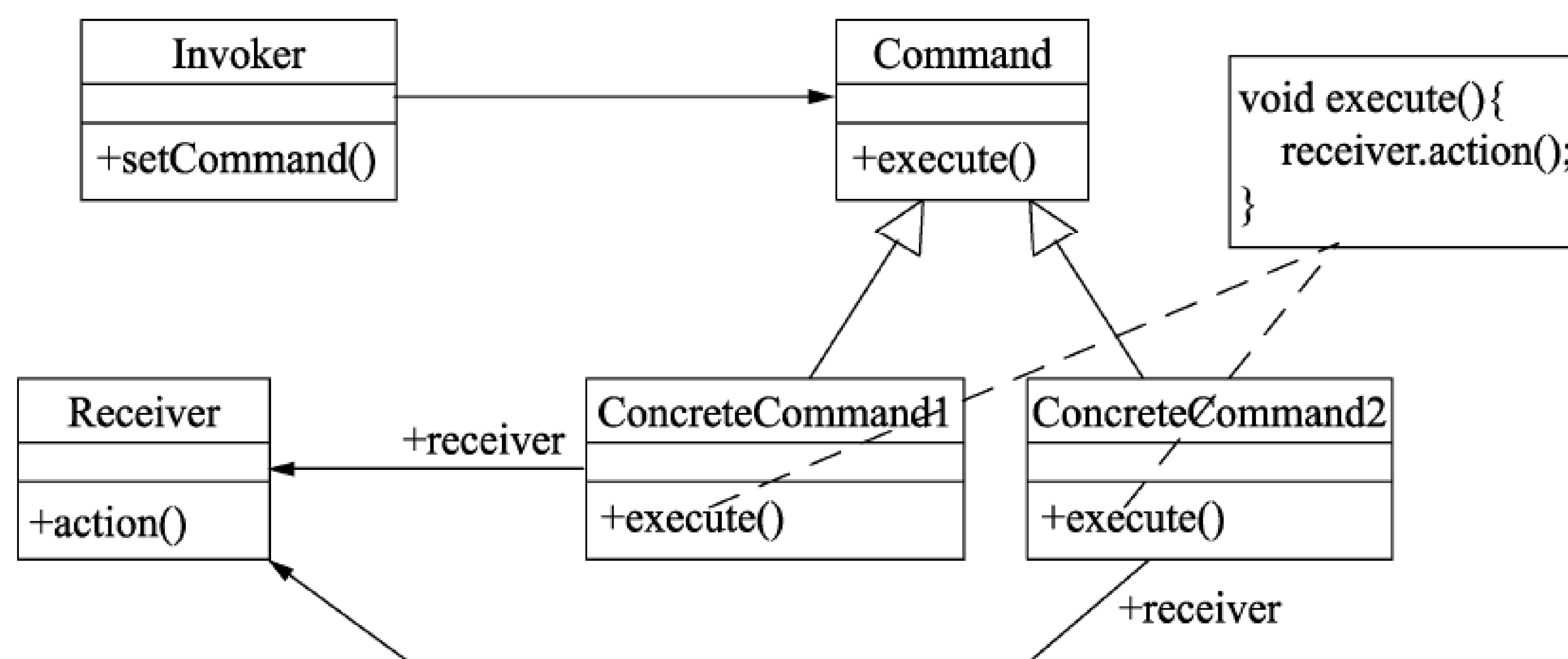


图 6-40　Command 模式类图

【Java 代码】

```
class Light{
public Light(){};
public Light(string name){/*代码省略*/}
public void on(){/*代码省略*/}  //开灯
public void off(){/*代码省略*/}  //关灯
};
(1) {
public void execute();
}
class LightOnCommand implements Command{//开灯命令
Light light;
public LightOnCommand(Light light){this.light=light;}
public void execute(){ (2) ;}
}
class LightOffCommand implements Command{//关灯命令
  Light light;
 public LightOffCommand(Light light){this.light=light;}
  public void execute(){ (3) ;}
}
class RemoteControl{ //遥控器
Command[] onCommands[7];
Command[] offCommands[7];
public RemoteControl() { /*代码省略*/ }
public void setCommand(int slot Command onCommand, Command offCommand) {
(4) =onCommand;
(5) =offCommand;
}
public void onButtonWasPushed(int slot) { (6);}
public void offButtonWasPushed(int slot) { (7); }
}
class remoteLoader {
public static void main (string[] args){
RemoteControl remoteControl = new RemoteControl();
Light livingRoomLight=new Light("Living Room");
Light kitchenLight=new Light("kitchen");
```

```
LightOnCommand livingRoomLightOn=new LightOnCommand(livingRoomLight);
LightOffCommand livingRoomLightOff=new LightOffCommand(livingRoomLight);
LightOnCommand kitchenLightOn=new LightOnCommand(kitchenLight);
LightOffCommand kitchenLightOff=new LightOffCommand(kitchenLight);
remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);
remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
remoteControl.onButtonWasPushed(0);
remoteControl.offButtonWasPushed(0);
remoteControl.onButtonWasPushed(1);
remoteControl.offButtonWasPushed(1);
}
}
```

解析：

本题考查设计模式的实现，难度较小。根据类图和已有代码可写出空缺的代码，第(1)空是 Command 接口的实现为 interface Command，(2)、(3)空分别执行开灯和关灯的操作，分别为 light.on()和 light.off()。(4)、(5)、(6)、(7)空为几种方法的实现，分别为 onCommands[slot]、offCommands[slot]、onCommands[slot].execute()和 offCommands[slot].execute()。

答案：

(1) interface Command。

(2) light.on()。

(3) light.off()。

(4) onCommands[slot]。

(5) offCommands[slot]。

(6) onCommands[slot].execute()。

(7) offCommands[slot].execute()。

例 10 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2015 年 5 月试题六)

【说明】

某图书管理系统中管理着两种类型的文献：图书和论文。现在要求统计所有馆藏文献的总页码(假设图书馆中有一本 540 页的图书和两篇各 25 页的论文，那么馆藏文献的总页码就是 590 页)。采用 Visitor(访问者)模式实现该要求，得到如图 6-41 所示的类图。

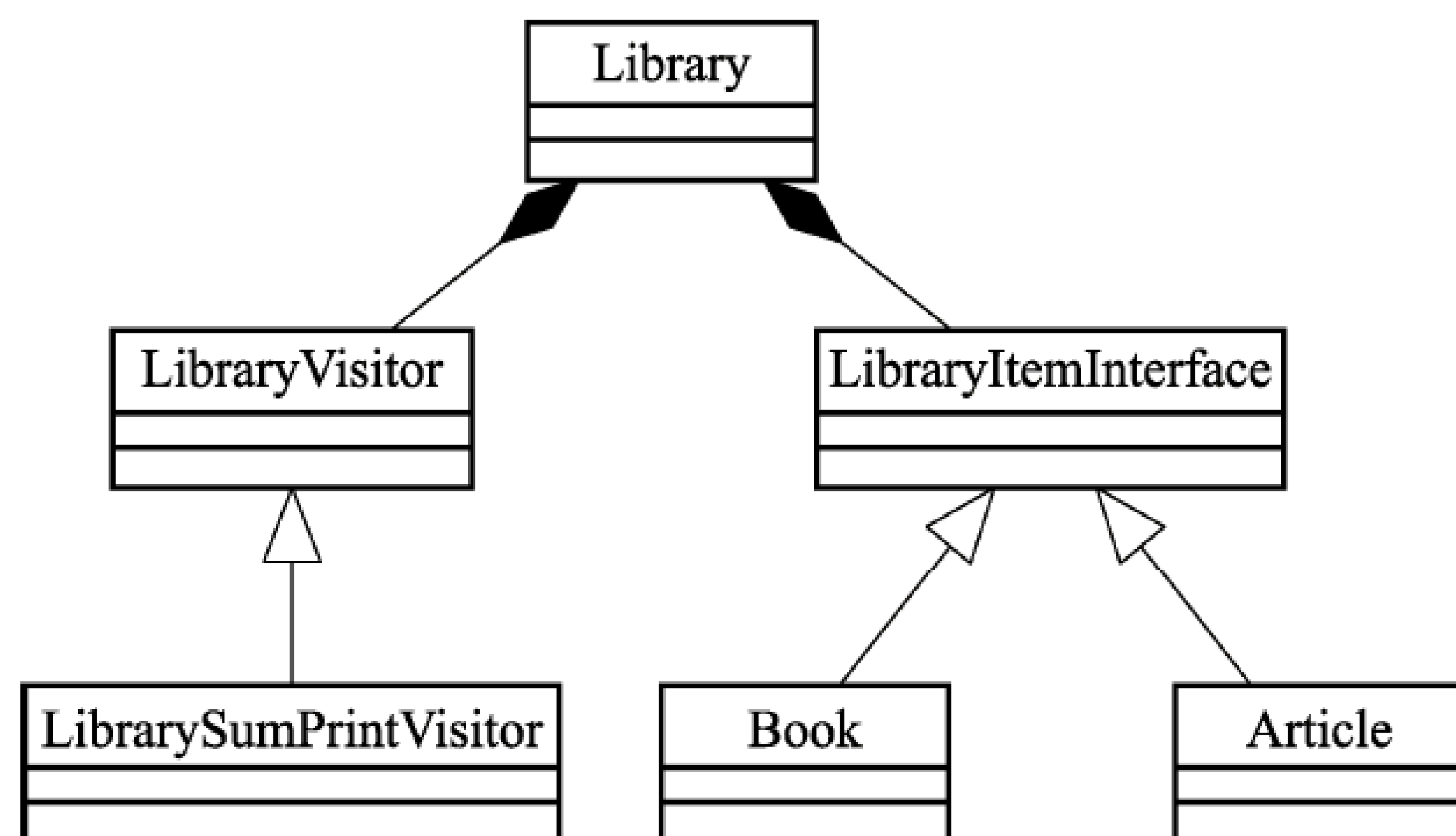
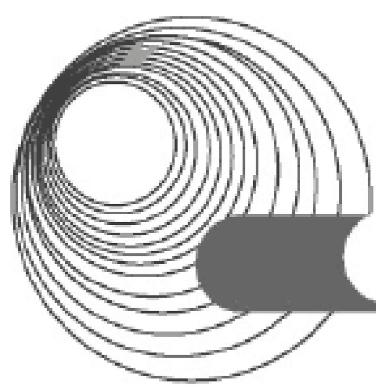


图 6-41 Visitor 模式类图

【Java 代码】

```
import java.util.*;
interface LibraryVisitor {
    (1)  ;
    (2)  ;
    void printSum();
}
class LibrarySumPrintVisitor implements LibraryVisitor {      //打印总页数
    private int sum = 0;
    public void visit(Book p_book) {
        sum = sum + p_book.getNumberOfPages();
        public void visit(Article p_article) {
        sum = sum + p_article.getNumberOfPages();
    }
    public void printSum(){
        System.out.println("SUM = " + sum);
    }
}
interface LibraryItemInterface {
     (3)  ;
}
class  Article implements LibraryItemInterface{
    private string m_title;      //论文名
    private string m_author;    //论文作者
    private int    m_start_page;
    private int    m_end_page;
    public Article(string p_author, string p_title, int p_start_page, int 
p_end_page){
        m_title=p_title;
        m_author=p_author;
        m_end_page=p_end_page;
    }
    public int getNumberOfPages(){
        return m_end_page - m_start_page;
    }
    public void accept(LibraryVisitor visitor){
        (4)   ;
    }
}
class Book implements LibraryItemInterface{
    private string m_title;      //书名
    private string m_author;     //书作者
    private int    m_pages;      //页数
    public Book(string p_author, string p_title,int p_pages){
        m_title=p_title;
        m_author=p_author;
        m_pages=p_pages;
    }
```




```
    public int getNumberOfPages(){
        return m_pages;
    }
    public void accept(LibraryVisitor visitor){
        ___(5)___;
    }
}
```

解析：

本题主要考查设计模式中的访问者模式的访问。访问者模式就是表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。访问者模式把数据结构和作用于结构上的操作之间的耦合解脱开，使得操作集合可以相对自由地演化。该模式的目的是要把处理从数据结构分离出来。访问者模式下，增加新的操作很容易，因为增加新的操作就意味着增加一个新的访问者。访问者模式将有关的行为集中到一个访问者对象中。

以上图 Visitor 模式类图为例，说明各个类的功能。

LibraryVisitor 类：抽象访问者角色，声明了两个访问操作，分别访问 Book 和 Article。

LibrarySumPrintVisitor 类：具体访问者角色，实现 LibraryVisitor 所声明的接口，也就是访问 Book 和 Article 的操作。

LibraryItemInterface 类：抽象被访问者角色，声明了一个接受操作，接受一个访问者对象。

Book 类和 Article 类：具体被访问者角色，实现了 LibraryItemInterface 类的接受操作。

LibraryVisitor 接口要声明两个访问操作的函数，分别访问 Book 类和 Article 类，再根据 LibrarySumPrintVisitor 类中给出的具体实现，可以推断出空(1)和空(2)中应填入 void visit(Book p_book)和 void visit(Article p_article)。

空(3)中应填入 LibraryItemInterface 接口对访问操作函数的声明，根据 Article 类和 Book 类的具体实现，可推断出空(3)应填入 void accept(Library Visitor visitor)。

空(4)和空(5)中应填入 Book 类和 Article 类对 accept 函数的具体实现，即 visitor.visit(this)和 visitor.visit(this)。

答案：

(1) void visit(Book p_book)。

(2) void visit(Article p_article)。

(3) void accept(Library Visitor visitor)。

(4) visitor.visit(this)。

(5) visitor.visit(this)。

例 11 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2015 年 11 月试题六)

【说明】

某大型购物中心欲开发一套收银软件，要求其能够支持购物中心在不同时期推出的各种促销活动，如打折、返利(例如，满 300 返 100)等。现采用策略(Strategy)模式实现该要求，得到如图 6-42 所示的类图。

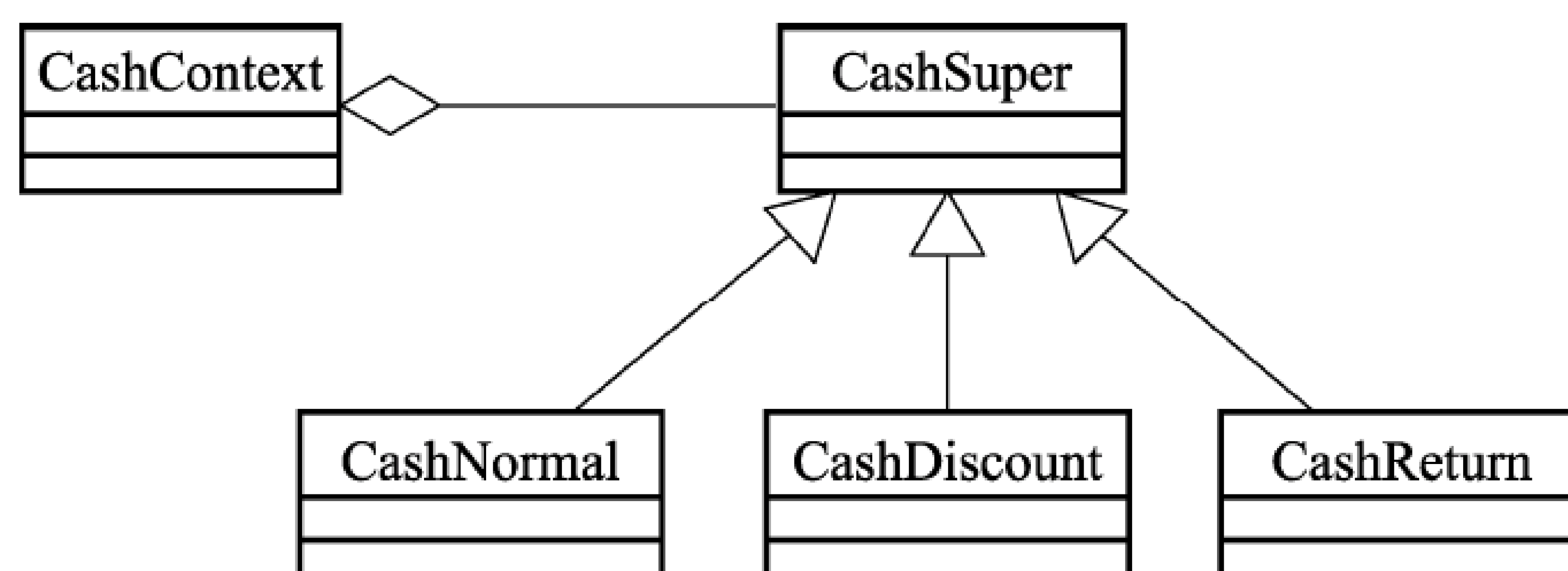
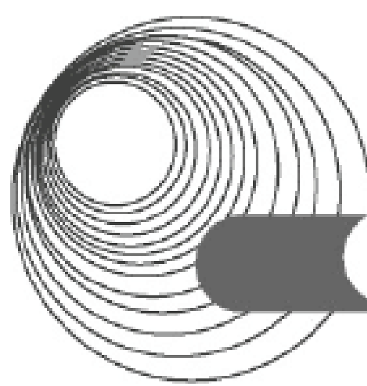


图 6-42　策略模式类图

【Java 代码】

```
import java.util.*;
enum TYPE { NORMAL, CASH_DISCOUNT, CASH_RETURN};
interface CashSuper {
   public  (1)  ;
}
class CashNormal implements CashSuper{    // 正常收费子类
   public double acceptCash(double money){
      return money;
   }
}
class CashDiscount implements CashSuper {
   private double moneyDiscount;                    // 折扣率
   public CashDiscount(double moneyDiscount) {
      this moneyDiscount = moneyDiscount;
   }
   public double acceptCash(double money) {
      return money* moneyDiscount;
   }
}
class CashReturn implements CashSuper {                // 满额返利
   private double moneyCondition;
   private double moneyReturn;
   public CashReturn(double moneyCondition, double moneyReturn) {
      this.moneyCondition=moneyCondition;     // 满额数额
      this.moneyReturn=moneyReturn;          // 返利数额
   }
   public double acceptCash(double money) {
      double result = money;
      if(money >= moneyCondition)
         result=money-Math.floor(money/moneyCondition)*moneyReturn;
      return result;
   }
}
class CashContext{
   private CashSuper cs;
   private TYPE  t;
   public CashContext(TYPE t)  {
      switch(t){
```




```
            case NORMAL:     // 正常收费
              ___(2)___ ;
              break;
            case CASH_DISCOUNT:      // 打 8 折
            ___(3)___;
              break;
            case CASH_RETURN:      // 满 300 返 100
            ___(4)___;
              break;
          }
       }
       public double GetResult(double money) {
          ___(5)___;
       }
       //此处略去 main()函数
    }
```

解析：

(1) 根据题意，本题使用的是策略模式，判断当前时期采用的是哪种促销策略，根据不同的促销策略执行不同的支付方式。CashSuper 定义了一个支付方式的接口，其余支付方式类均继承自 CashSuper 这个接口，从下面定义的正常收费子类(CashNormal)、折扣率类(CashDiscount)、满额返利类(CashReturn)可以看出，它们实现了共同的 CashSuper 接口的方法，即 acceptCash 函数。所以空(1)中应填写 acceptCash 函数的定义。

(2) CashContext 类是一个收费的实体类，在该类中，通过 TYPE 类型的变量 t 来判断当前的促销策略，当促销策略不同时，使用不同的支付类实例来调用 acceptCash 方法。因此，当促销策略是 NORMAL 时，应创建 CashNormal 类的实例，空(2)中应该填入 cs = new CashNormal()。

(3) 当促销策略是 CASH_DISCOUNT 时，应创建 CashDiscount 类的实例，空(3)中应填入 cs = new CashDiscount(0.8)。

(4) 当促销策略是 CASH_RETURN 时，应创建 CashReturn 类的实例，空(4)中应填入 cs = new CashReturn(300,100)。

(5) GetResult 函数，返回在当前支付方式下计算的结果，经过(2)~(4)步骤，已经根据促销策略创建好了相应的支付方式类，所以，只要调用 cs. acceptCash(money)就可返回结果。

答案：

(1) double acceptCash(double money)。

(2) cs = new CashNormal()。

(3) cs = new CashDiscount(0.8)。

(4) cs = new CashReturn(300,100)。

(5) return cs.acceptCash(money)。

6.2.3　同步练习

1. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2009 年 11 月试题六)

【说明】

现欲构造一文件/目录树，采用组合(Composite)设计模式来设计，得到的类图如图 6-43 所示。

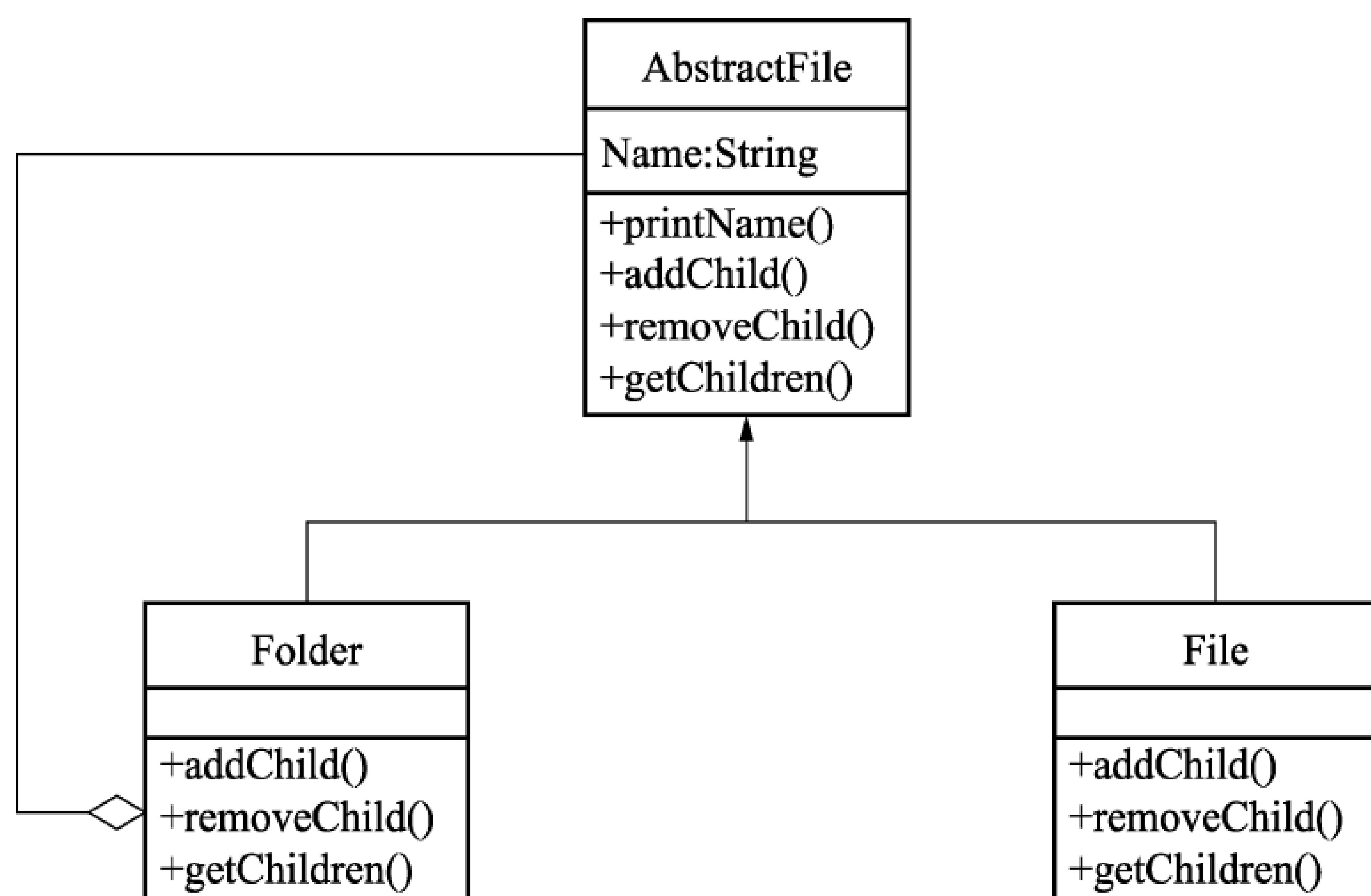


图 6-43　第 1 题类图

【Java 程序】

```
import java.util.ArrayList;
import java.util.List;
  (1)  class AbstractFile
 {
      protected String name;
      public void printName(){System.out.println(name);}
      public abstract boolean addChild(AbstractFile file);
      public abstract boolean removeChild(AbstractFile file);
      public abstract List<AbstractFile> getChildren();
}
class File extends AbstractFile
{
    public File(String name){this.name=name;}
    public boolean addChild(AbstractFile file){return false;}
    public boolean removeChild(AbstractFile file){return false;}
    public List<AbstractFile> getChildren(){return  (2)  ;}
}
class Folder extends AbstractFile{
      private List <AbstractFile> childList;
```

```
        public Folder(String name) {
            this.name=name;
            this.childList=new ArrayList<AbstractFile>();
        }
        Public  boolean addChild(AbstractFile file) {return childList.add(file);}
        Public  boolean removeChild(AbstractFile file) {return childList.remove(file);}
        public  (3)  <AbstractFile> getChildren() {return  (4)  ;}
    }
    public class Client {
        public static void main(String[] args) {
          //构造一个树型的文件/目录结构
            AbstractFile rootFolder= new Folder("c:\\ ");
            AbstractFile compositeFolder=new Folder("composite");
            AbstractFile windowsFolder=new Folder("windows");
            AbstractFile file=new File("TestComposite.java");
            rootFolder.addChild(compositeFolder);
            rootFolder.addChild(windowsFolder);
            compositeFolder.addChild(file);
           //打印目录文件树
            printTree(rootFolder);
        }
       private static void printTree(AbstractFile ifile) {
            ifile.printName();
            List <AbstractFile> children=ifile.getChildren;
            if(children==null) return;
            for (AbstractFile ifile.children) {
                  (5)  ;
            }
        }
    }
```

该程序运行后输出结果为：

```
c:\
composite
TestComposite.java
Windows
```

2. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2009 年 5 月试题七)

【说明】

现欲实现一个图像浏览系统，要求该系统能够显示 BMP、JPEG 和 GIF 三种格式的文件，并且能够在 Windows 和 Linux 两种操作系统上运行。系统首先将 BMP、JPEG 和 GIF 三种格式的文件解析为像素矩阵，然后将像素矩阵显示在屏幕上。系统需具有较好的扩展性以支持新的文件格式和操作系统。为满足上述需求并减少所需生成的子类数目，采用桥接(Bridge)设计模式进行设计，所得类图如图 6-44 所示。

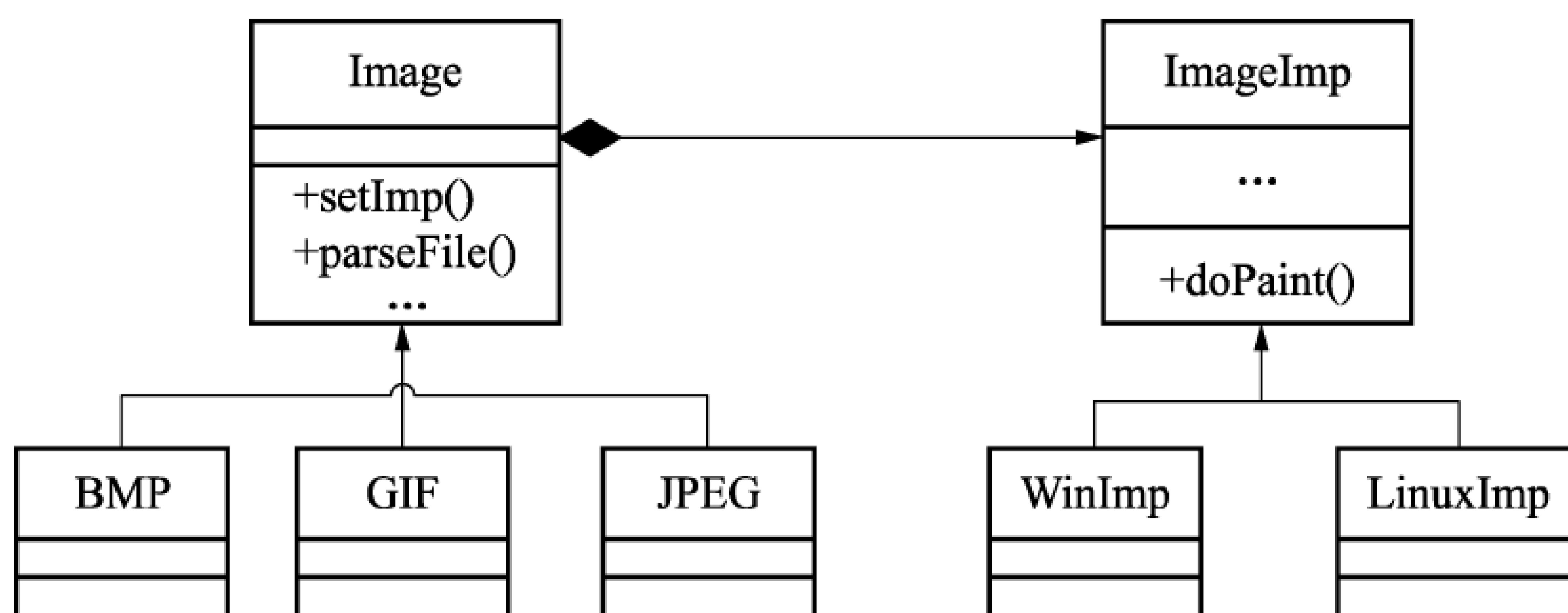


图 6-44　第 2 题类图

采用该设计模式的原因在于：系统解析 BMP、GIF 与 JPEG 文件的代码仅与文件格式相关，而在屏幕上显示像素矩阵的代码则仅与操作系统相关。

【Java 程序】

```
class Matrix{    //各种格式的文件最终都被转化为像素矩阵
    //此处代码省略
};
abstract class ImageImp{
    public  abstract void doPaint(Matrix m);     //显示像素矩阵 m
};
class WinImp extends ImageImp{
    public void doPaint(Matrix m){ /*调用 Windows 系统的绘制函数绘制像素矩阵*/ }
};
class LinuxImp extends ImageImp{
    public void doPaint(Matrix m){/*调用 Linux 系统的绘制函数绘制像素矩阵*/}
};
abstract class Image
{
    public void setImp(ImageImp imp)
    {
      ___(1)___= imp;
    }
    public abstract void parseFile(String fileName);
    protected ___(2)___imp;
};
class BMP extends Image
{
    public  void parseFile(String fileName)
    {
    //此处解析 BMP 文件并获得一个像素矩阵对象 m
    ___(3)___ ;// 显示像素矩阵 m
    }
};
class GIF extends Image
{
    //此处代码省略
};
```

```
class JPEG extends Image
{
    //此处代码省略
};
public class javaMain
{
    public static void main(String[] args)
    {
    //在 Windows 操作系统上查看 demo.bmp 图像文件
        Image image1 = (4) ;
        ImageImp imageImp1 = (5) ;
       (6) ;
       image1.parseFile("demo.bmp");
    }
}
```

现假设该系统需要支持 10 种格式的图像文件和 5 种操作系统，不考虑类 Matrix 和类 javaMain，若采用桥接设计模式，则至少需要设计 (7) 个类。

3. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2008 年 12 月试题七)

【说明】

已知某类库开发商提供了一套类库，类库中定义了 Application 类和 Document 类，它们之间的关系如图 6-45 所示。其中，Application 类表示应用程序自身，而 Document 类则表示应用程序打开的文档。Application 类负责打开一个已有的以外部形式存储的文档，如一个文件，一旦从该文件中读出信息后，它就由一个 Document 对象表示。

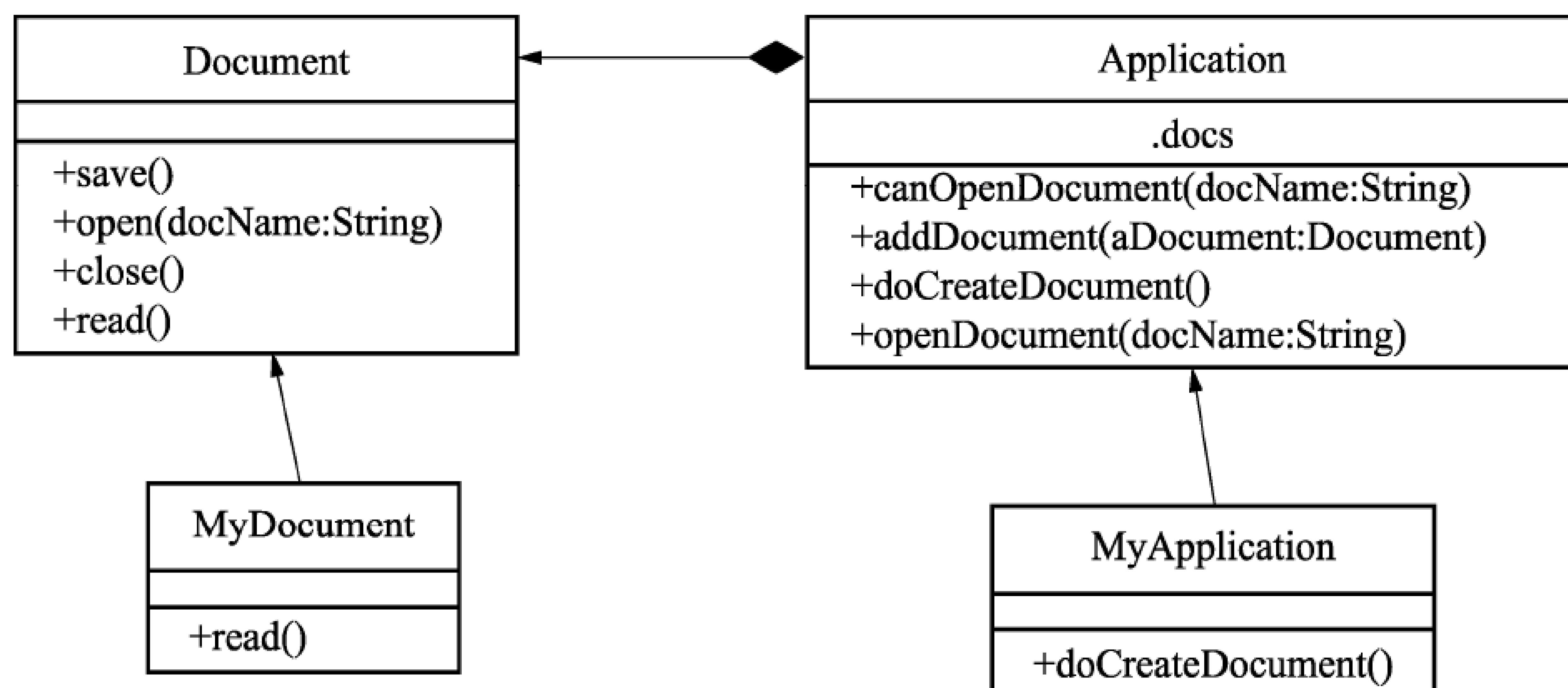


图 6-45 Application 与 Document 的关系

当开发一个具体的应用程序时，开发者需要分别创建自己的 Application 和 Document 子类，如图 6-45 中的类 MyApplication 和类 MyDocument，并分别实现 Application 和 Document 类中的某些方法。

已知 Application 类中的 openDocument 方法采用了模板方法(Template Method)设计模式，该方法定义了打开文档的每一个主要步骤，具体如下。

(1) 首先检查文档是否能够被打开，若不能打开，则给出出错信息并返回。
(2) 创建文档对象。
(3) 通过文档对象打开文档。
(4) 通过文档对象读取文档信息。
(5) 将文档对象加入到 Application 的文档对象集合中。

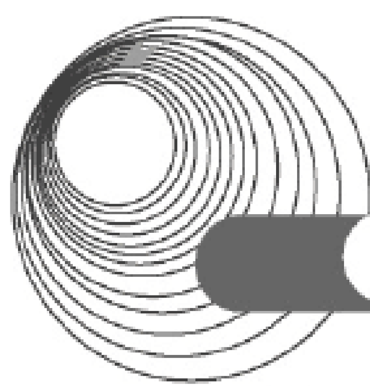
【Java 程序】

```
abstract class Document
{
    public void save(){ /*存储文档数据，此处代码省略*/ }
    public void open(String docName){ /*打开文档，此处代码省略*/ }
    public void close(){ /*关闭文档，此处代码省略*/ }
    public abstract void read(String docName);
};
abstract class Appplication
{
    private Vector <(1)> docs; /*文档对象集合*/
    public boolean canOpenDocument(String docName){
        /*判断是否可以打开指定文档，返回真值表示可以打开，返回假值表示不可打开，此处代码省略*/
    }
    public void addDocument(Document aDocument)
    {
        /*将文档对象添加到文档对象集合中*/
        docs.add((2));
    }
    public abstract Document doCreateDocument(); /*创建一个文档对象*/
    public void openDocument(String docName){ /*打开文档*/
        if ((3))
        {
            System.out.println( "文档无法打开！" );
            return;
        }
    (4) adoc =(5);
    (6) ;
    (7) ;
    (8) ;
    }
};
```

4. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2008 年 5 月试题七)

【说明】

已知某企业欲开发一家用电器遥控系统，即用户使用一个遥控器即可控制某些家用电器的开与关。遥控器如图 6-46 所示。该遥控器共有 4 个按钮，编号分别是 0~3，按钮 0 和按钮 2 能够遥控打开电器 1 和电器 2，按钮 1 和按钮 3 则能遥控关闭电器 1 和电器 2。由于遥控系统需要支持形式多样的电器，因此该遥控系统的设计要求具有较高的扩展性。现假



设需要控制客厅电视和卧室电灯，对该遥控系统进行设计所得类图如图 6-47 所示。

在图 6-47 中，类 RomoteController 的方法 onPressButton(int button)表示当遥控器按键按下时调用的方法，参数为按键的编号；Command 接口中 on 和 off 方法分别用于控制电器的开与关；Light 中 turnLight(int degree)方法用于调整电灯光的强弱，参数 degree 值为 0 时表示关灯，值为 100 时表示开灯，并且将灯光亮度调整到最大；TV 中 setChannel(int channel)方法表示设置电视播放的频道，参数 channel 值为 0 时表示关闭电视，为 1 时表示开机并将频道切换为第 1 频道。

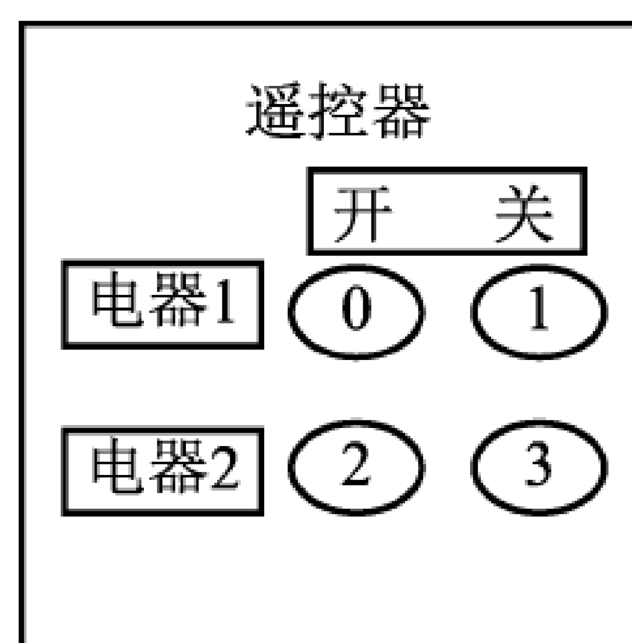


图 6-46 遥控器

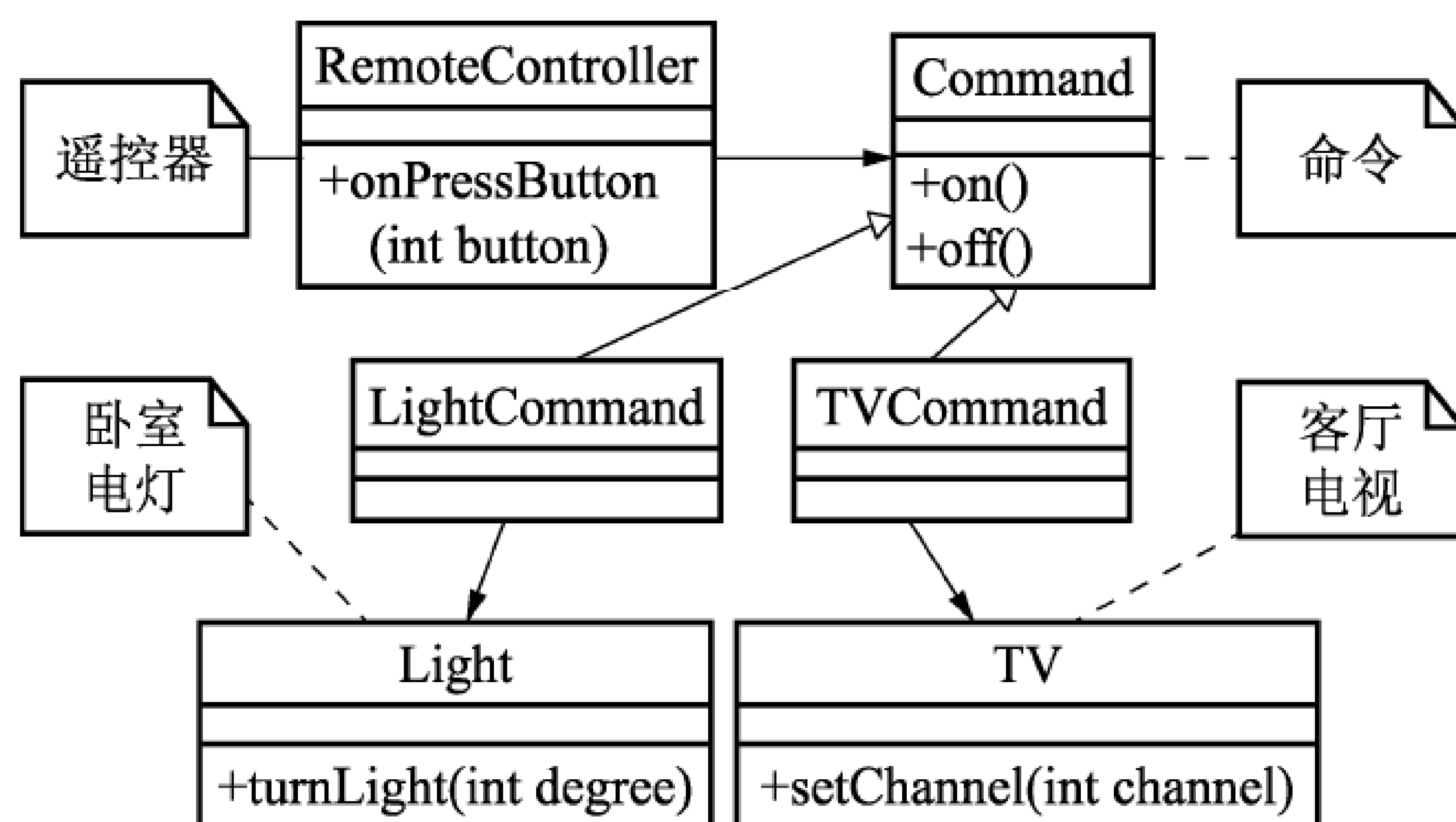


图 6-47 设计类图

【Java 程序】

```
class Light                          //电灯类
{
    public void turnLight(int degree){ //调整灯光亮度，0 表示关灯，100 表示亮度
最大}
};
class TV                           //电视机类
{
    public void setChannel(int channel){//0 表示关机，1 表示开机并切换到 1 频道}
};
interface Command         //抽象命令类
{
    void on();
    void off();
};
class RemoteController     //遥控器类
{
    protected Command []commands = new Command[4];
    //遥控器有 4 个按钮，按照编号分别对应 4 个 Command 对象
    public void onPressButton(int button)
    {
    //按钮被按下时执行命令对象中的命令
        if(button % 2 == 0)commands[button].on();
        else commands[button].off();
    }
    public void setCommand(int button, Command command)
```

```
        {
        (1) = command; //设置每个按钮对应的命令对象
        }
    };
    class LightCommand implements Command   //电灯命令类
    {
        protected Light light; //指向要控制的电灯对象
        public void on(){light.turnLight(100);};
        public void off(){light. (2) ;};
        public LightCommand(Light light){this.light = light;};
    };
    class TVCommand implements Command    //电视机命令类
    {
        protected TV tv; //指向要控制的电视机对象
        public void on(){tv. (3) ;};
        public void off(){tv.setChannel(0);};
        public TVCommand(TV tv){this.tv = tv;};
    };
    public class rs
    {
        public static void main(String[] args){
        Light light = new Light(); TV tv = new TV();//创建电灯和电视对象
        LightCommand lightCommand = new LightCommand(light);
        TVCommand tvCommand = new TVCommand(tv);
        RemoteController remoteController = new RemoteController();
        //设置按钮和命令对象
        remoteController.setCommand(0, (4) );

        …//此处省略设置按钮 1、按钮 2 和按钮 3 的命令对象代码
        }
    }
```

本题中，应用命令模式能够有效地让类 (5) 和类 (6) 、类 (7) 之间的耦合性降至最小。

5. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2007 年 11 月试题七)

【说明】

已知某企业的采购审批是分级进行的，即根据采购金额的不同由不同层次的主管人员来审批，主任可以审批 5 万元以下(不包括 5 万元)的采购单，副董事长可以审批 5 万~10 万元(不包括 10 万元)的采购单，董事长可以审批 10 万~50 万元(不包括 50 万元)的采购单，50 万元及以上的采购单就需要开会讨论决定。

采用责任链设计模式(Chain of Responsibility)对上述过程进行设计后得到的类图如图 6-48 所示。

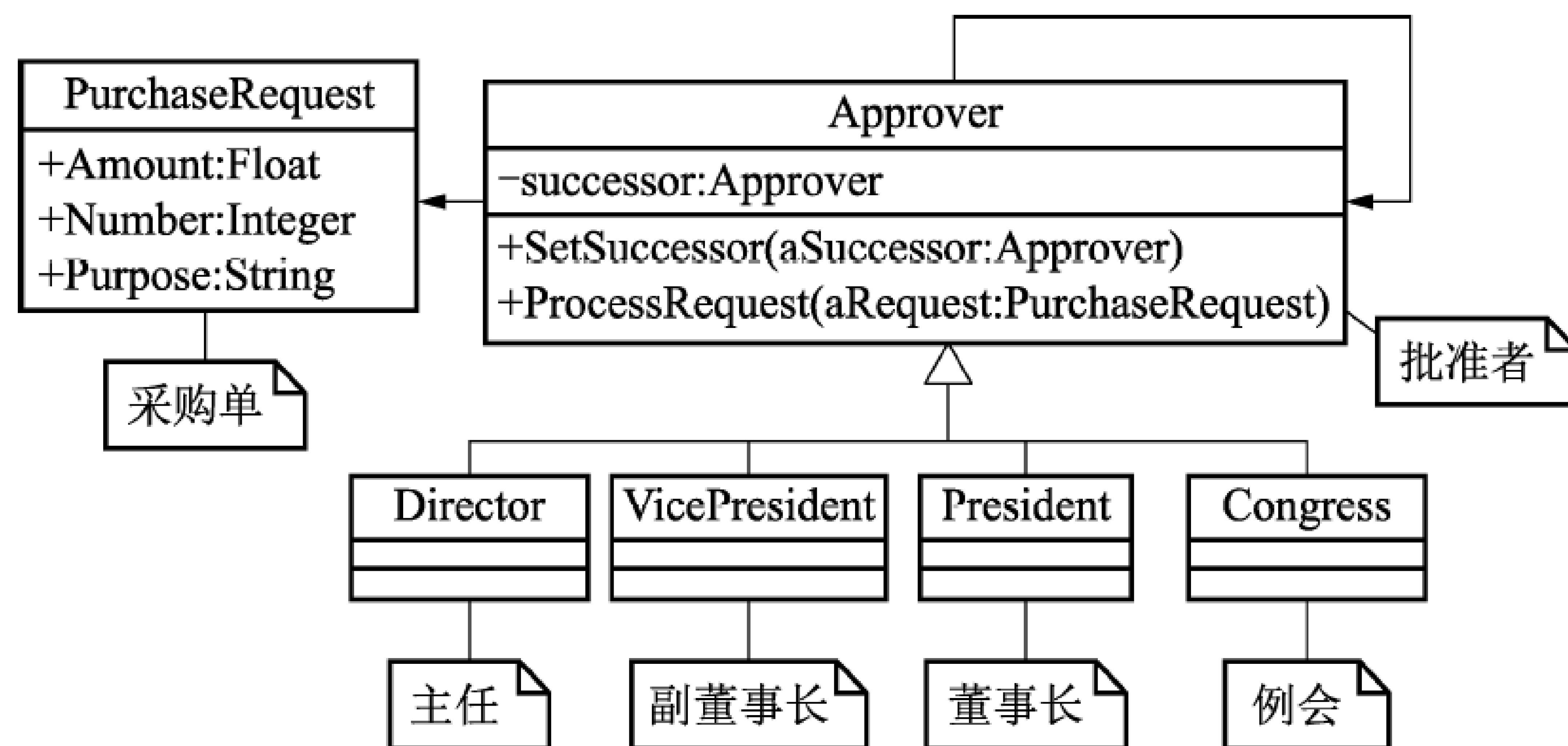


图 6-48　设计类图

【Java 程序】

```
class PurchaseRequest
{
    public  double Amount;      // 一个采购的金额
    public  int Number;         // 采购单编号
    public  string Purpose;     // 采购目的
    };
class Approver                  // 审批者类
{
    public Approver(){successor = null;}
    public void ProcessRequest(PurchaseRequest aRequest)
    {
        if (successor != null){ successor.___(1)___;   }
    }
    public void SetSuccessor(Approver aSuccessor){ successor = aSuccessor; }
    private ___(2)___ successor;
};

class Congress extends Approver
{
    public  void ProcessRequest(PurchaseRequest aRequest){
        if(aRequest.Amount >= 500000){  /* 决定是否审批的代码省略 */    }
        else ___(3)___.ProcessRequest(aRequest);
    }
};
class Director extends Approver
{
    public void ProcessRequest(PurchaseRequest aRequest)
    { /* 此处代码省略 */    }
};
class President extends Approver
{
    public void ProcessRequest(PurchaseRequest aRequest)
    { /* 此处代码省略 */ }
```

```
};
class VicePresident extends Approver
{
  public void ProcessRequest(PurchaseRequest aRequest)
{ /* 此处代码省略 */ }
};

public class rs
{
    public static void main(String[] args) throws IOException
    {
        Congress Meeting = new Congress();
        VicePresident Sam = new VicePresident();
        Director Larry = new Director();
        President Tammy = new President();
        // 构造责任链
        Meeting.SetSuccessor(null); Sam.SetSuccessor(  (4)  );
        Tammy.SetSuccessor(  (5)  ); Larry.SetSuccessor(  (6)  );
        // 构造一采购审批请求
        PurchaseRequest aRequest = new PurchaseRequest();
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        aRequest.Amount = Double.parseDouble(br.readLine());
          (7)  .ProcessRequest(aRequest);     // 开始审批
        return;
    }
}
```

6. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2007 年 5 月试题七)

【说明】

某游戏公司现欲开发一款面向儿童的模拟游戏，该游戏主要模拟现实世界中各种鸭子的发声特征、飞行特征和外观特征。游戏需要模拟的鸭子种类及其特征如表 6-9 所示。

表 6-9　游戏需要模拟的鸭子种类及特征

鸭子种类	发声特征	飞行特征	外观特征
灰鸭(MallardDuck)	发出“嘎嘎”声(Quack)	用翅膀飞行(FlyWithWings)	灰色羽毛
红头鸭(RedHeadDuck)	发出“嘎嘎”声(Quack)	用翅膀飞行(FlyWithWings)	灰色羽毛、头部红色
棉花鸭(CottonDuck)	不发声(QuackNoWay)	不能飞行(FlyNoWay)	白色
橡皮鸭(RubberDuck)	发出橡皮与空气摩擦的声音(Squeak)	不能飞行(FlyNoWay)	黑白橡皮色

为支持将来能够模拟更多种类鸭子的特征，采用策略设计模式(Strategy)设计的类图如图 6-49 所示。

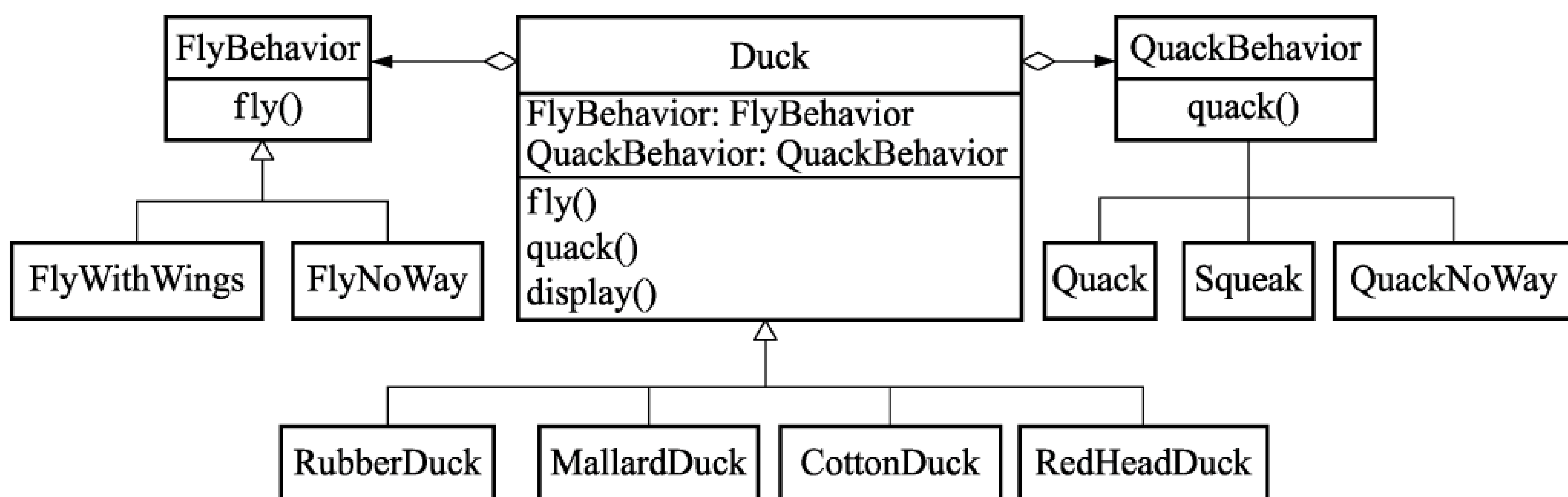


图 6-49　类图

其中，Duck 为抽象类，描述了抽象的鸭子，而类 RubberDuck、MallardDuck、CottonDuck 和 RedHeadDuck 分别描述具体的鸭子种类，方法 fly()、quack()和 display()分别表示不同种类的鸭子都具有飞行特征、发声特征和外观特征；接口 FlyBehavior 与 QuackBehavior 分别用于表示抽象的飞行行为与发声行为；类 FlyNoWay 与 FlyWithWings 分别描述不能飞行的行为和用翅膀飞行的行为；类 Quack、Squeak 与 QuackNoWay 分别描述发出“嘎嘎”声的行为、发出橡皮与空气摩擦声的行为与不发声的行为。请填补以下代码中的空缺。

【Java 程序】

```
__(1)__ FlyBehavior
{
    public void fly( );
};
__(2)__ QuackBehavior
{
    public void quack( );
};
class FlyWithWings implements FlyBehavior
{
    public void fly( ){System.out.println("使用翅膀飞行! ");}
};
class FlyNoWay implements FlyBehavior{
    public void fly( ){System.out.println("不能飞行! ");}
};
class Quack implements QuackBehavior
{
    public void quack( ){System.out.println("发出\'嘎嘎'\声! ");}
};
class Squeak implements QuackBehavior
{
    public void quack( ){System.out.println("发出空气与橡皮摩擦声! ");}
};
class QuackNoWay implements QuackBehavior
{
    public void quack( ){System.out.println("不能发声! ");}
};
Abstract class Duck
{
    protected  FlyBehavior __(3)__;
```

```
    protected  QuackBehavior  (4) ;
    public  void fly( ){ (5) ;};
    public  void quack( ){ (6) ; };
    public  (7)  void display( );
};
class RubberDuck extends Duck
{
    public RubberDuck( ){
            flyBehavior=new  (8) ;
            quackBehavior=new  (9) ;
    }
public void display( ){/*此处省略显示橡皮鸭的代码*/}
};
//其他代码省略
```

7. 阅读以下说明及 Java 程序，将应填入(n)处的子句写在答题纸的对应栏内。(2006 年 11 月试题七)

【说明】

传输门是传输系统中的重要装置。传输门具有 Open(打开)、Closed(关闭)、Opening(正在打开)、StayOpen(保持打开)、Closing(正在关闭)5 种状态。触发状态的转换事件有 click、complete 和 timeout 3 种。事件与其相应的状态转换如图 6-50 所示。

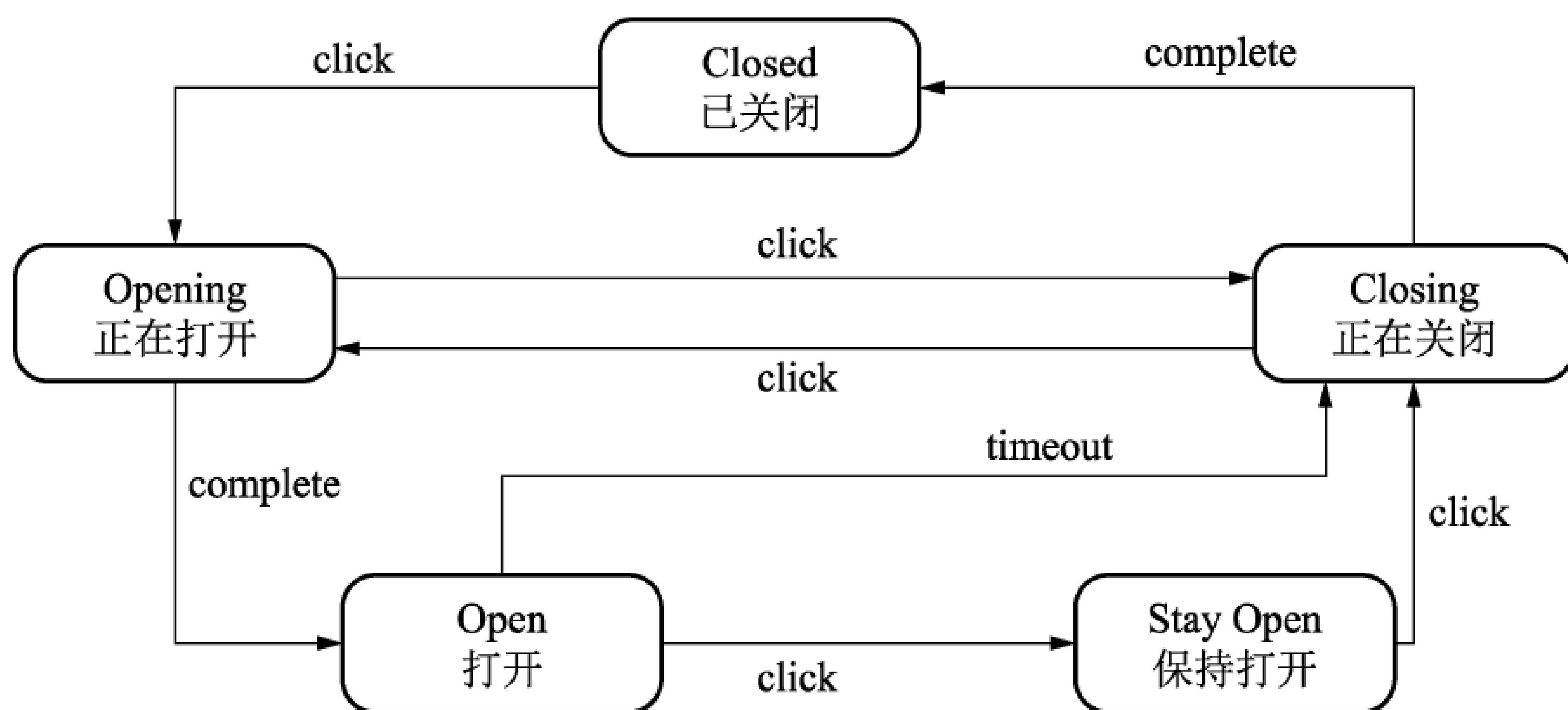


图 6-50　传输门响应事件与其状态转换图

下面的 Java 程序 1 与 Java 程序 2 分别用两种不同的设计思路对传输门进行状态模拟，请填补代码中的空缺。

【Java 程序 1】

```
public class Door
{
     public static final int CLOSED = 1;
     public static final int OPENING = 2;
     public static final int OPEN = 3;
     public static final int CLOSING = 4;
     public static final int STAYOPEN = 5;
     private int state = CLOSED;
```

```
    //定义状态变量，用不同整数表示不同状态
    private void setState(int state){this.state = state;}
    //设置传输门当前状态
    public void getState()
    {
      //此处代码省略，本方法输出状态字符串
      //例如，当前状态为 CLOSED 时，输出字符串为“CLOSED”
    }
    public void click()       //发生 click 事件时进行状态转换
    {
        if ( (1) ) setState(OPENING);
        else if ( (2) ) setState(CLOSING);
        else if ( (3) state==OPEN) setState(STAYOPEN);
                        //发生 timeout 事件时进行状态转换
        public void timeout(){ if (state == OPEN) setState(CLOSING); }
        public void complete()  //发生 complete 事件时进行状态转换
        {
            if (state == OPENING) setState(OPEN);
            else if (state == CLOSING) setState(CLOSED);
        }
    }
    public static void main(String[] args)
    {
         Door aDoor = new Door();
         aDoor.getState(); aDoor.click(); aDoor.getState(); aDoor.complete();
         aDoor.getState(); aDoor.click(); aDoor.getState(); aDoor.click();
         aDoor.getState(); return;
    }
  }
```

【Java 程序 2】

```
public class Door
{
        public final DoorState CLOSED = new DoorClosed(this);
        public final DoorState OPENING = new DoorOpening(this);
        public final DoorState OPEN = new DoorOpen(this);
        public final DoorState CLOSING = new DoorClosing(this);
        public final DoorState STAYOPEN = new DoorStayOpen(this);
        private DoorState state = CLOSED;
        //设置传输门当前状态
        public void setState(DoorState state){ this.state = state;}
        public void getState(){  //根据当前状态输出对应的状态字符串
           System.out.println(state.getClass().getName());
        }
        public void click(){ (4) ;}  //发生 click 事件时进行状态转换
        public void timeout(){ (5) ;}  //发生 timeout 事件时进行状态转换
        public void complete(){ (6) ;}  //发生 complete 事件时进行状态转换
        public static void main(String[] args)
        {
```

```
            Door aDoor = new Door();
            aDoor.getState(); aDoor.click(); aDoor.getState(); aDoor.complete();
            aDoor.getState(); aDoor.timeout(); aDoor.getState(); return;
        }
    }
    public abstract class DoorState       //定义所有状态类的基基类
    {
            protected Door door;
            public DoorState(Door door) {this.door = door;}
            public void click() {}
            public void complete() {}
            public void timeout() {}
    }
    class DoorClosed extends DoorState      //定义一个基本的 Closed 状态
    {
          public DoorClosed(Door door) { super(door); }
          public void click() { (7) ; }

          //该类定义的其余代码省略
    }
        //其余代码省略
```

8. 阅读下列说明、图及 Java 程序，将应填入(n)处的子句写在答题纸的对应栏内。(2006 年 5 月试题七)

【说明】

某订单管理系统的部分 UML 类图如图 6-51 所示。

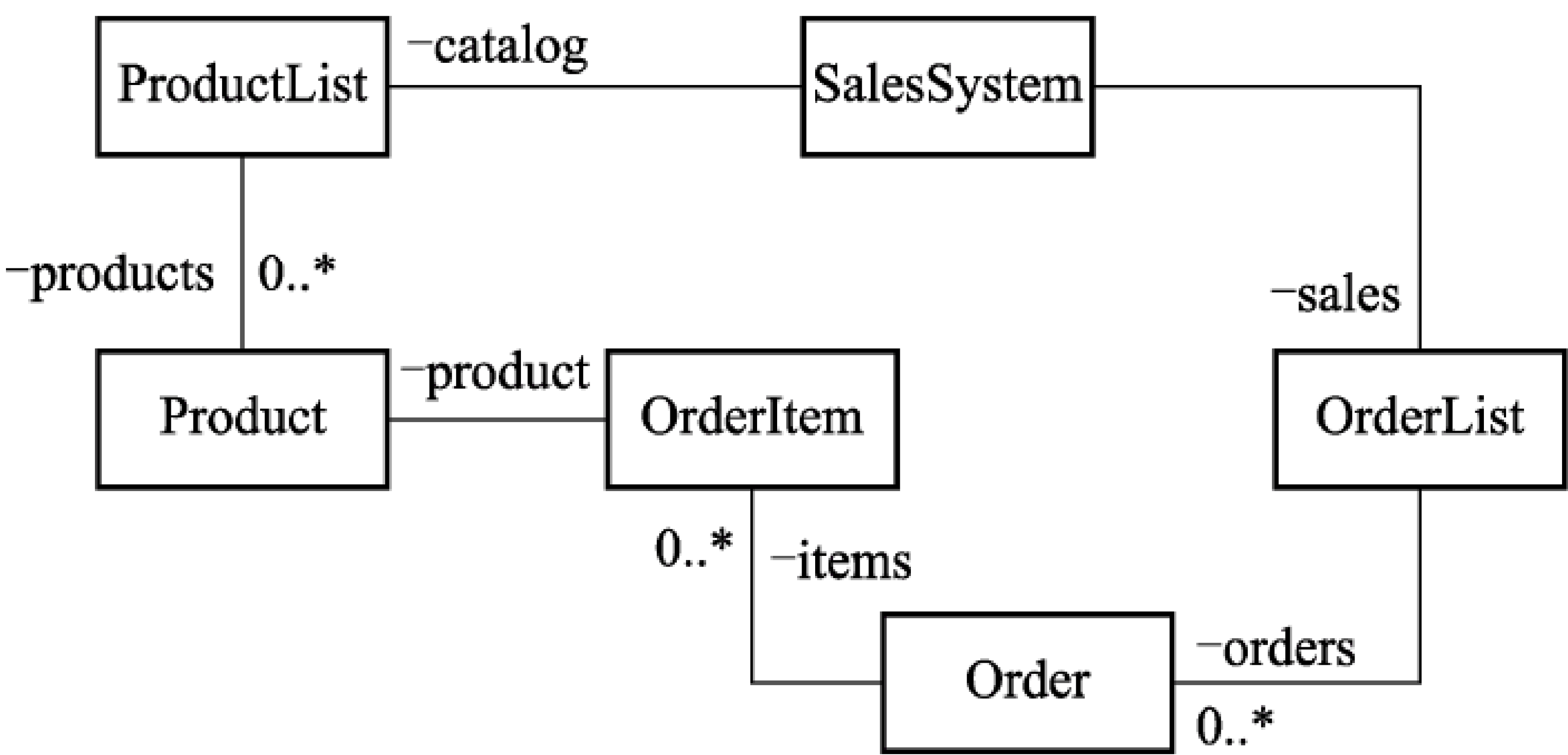


图 6-51 UML 类图

在图 6-51 中，Product 表示产品，ProductList 表示所销售产品的列表，Order 表示产品订单，OrderItem 表示产品订单中的一个条目，OrderList 表示订单列表，SalesSystem 提供订单管理系统的操作接口。各个类的部分属性和方法说明如表 6-10 所示。

表 6-10　各个类的部分属性和方法说明

类	成　员	说　明
ProductList	ArrayList<Product> products	
Product	String code	产品编号
	String description	产品描述
	double price	产品单价
	Boolean equals(Object object)	若两个产品相同则返回 true，否则返回 false
OrderItem	Product product	订单项中的产品
	int quantity	产品的订购数量
	Product getProduct()	获取订单项中的产品
Order	ArrayList<OrderItem> items	订单中包含的订单项
OrderList	ArrayList<Order> orders	订单
	void addOrder(Order order)	向订单列表中添加新订单
	int getNumberOfOrders()	获取订单列表中的订单总数
SalesSystem	ProductList catalog	产品目录
	OrderList sales	订单列表
	void statistic()	依次统计产品目录中每个产品的订购总量，并打印出每个产品的编号、说明、订购总量和订购金额

可以使用类 java.util.ArrayList<E>来实现对象的聚集关系，如图 6-51 中 OrderList 与 Order 之间的聚集关系。

for-each 循环提供了一种遍历对象集合的简单方法。在 for-each 循环中，可以指定需要遍历的对象集合以及用来接受集合中每个元素的变量，其语法格式如下：

```
for(用来接受集合中元素的变量：需要遍历的对象集合)
```

如果要使用 for-each 循环来遍历对象集合，那么包含该对象集合的类必须实现接口 java.util.Iterable<T>。

下面的 Java 程序 1 和 Java 程序 2 分别给出了类 OrderList 和方法 statistic 的 Java 代码。

【Java 程序 1】

```
import java.util.*;
public class OrderList  (1) 
{
    private ArrayList<Order> orders;
    public OrderList()
    {
        this.orders = new ArrayList<Order>();
    }
    public void addOrder(Order order)
    {
        this.orders.add(order);
    }
```

```
    public Iterator<Order> iterator()
    {
        return  (2) ;
    }
    public int getNumberOfOrders()
    {
        return this.orders.size();
    }
}
```

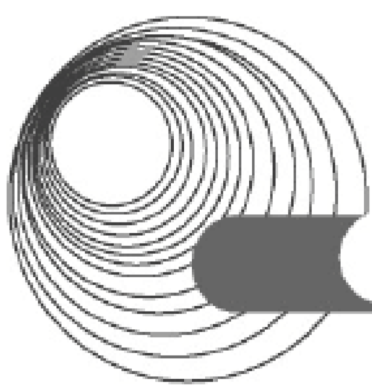
【Java 程序 2】

```
import java.util.*;
public class SalesSystem
{
    private ProductList catalog;
    private OrderList sales;
    private static PrintWriter stdOut = new PrintWriter(System.out, true);
    public void statistic()
    {
        for (Product product :  (3) )
        {
            int number = 0;
            for (Order order :  (4) )
            {
                for ( (5) : order)
                {
                    if (product.equals(item.getProduct()))
                    number += item.getQuantity();
                }
            }
            stdOut.println(product.getCode() + "  "
                + product.getDescription() + "  "
                + number + "  " + number * product.getPrice());
        }
    }
    //其余的方法未列出
}
```

9. 阅读下列说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。(2016 年 5 月试题六)

【说明】

某软件系统中，已设计并实现了用于显示地址信息的类 Address(如图 6-52 所示)，现要求提供基于 Dutch 语言的地址信息显示接口。为了实现该要求并考虑到以后可能还会出现新的语言的接口，决定采用适配器(Adapter)模式实现该要求，得到如图 6-52 所示的类图。



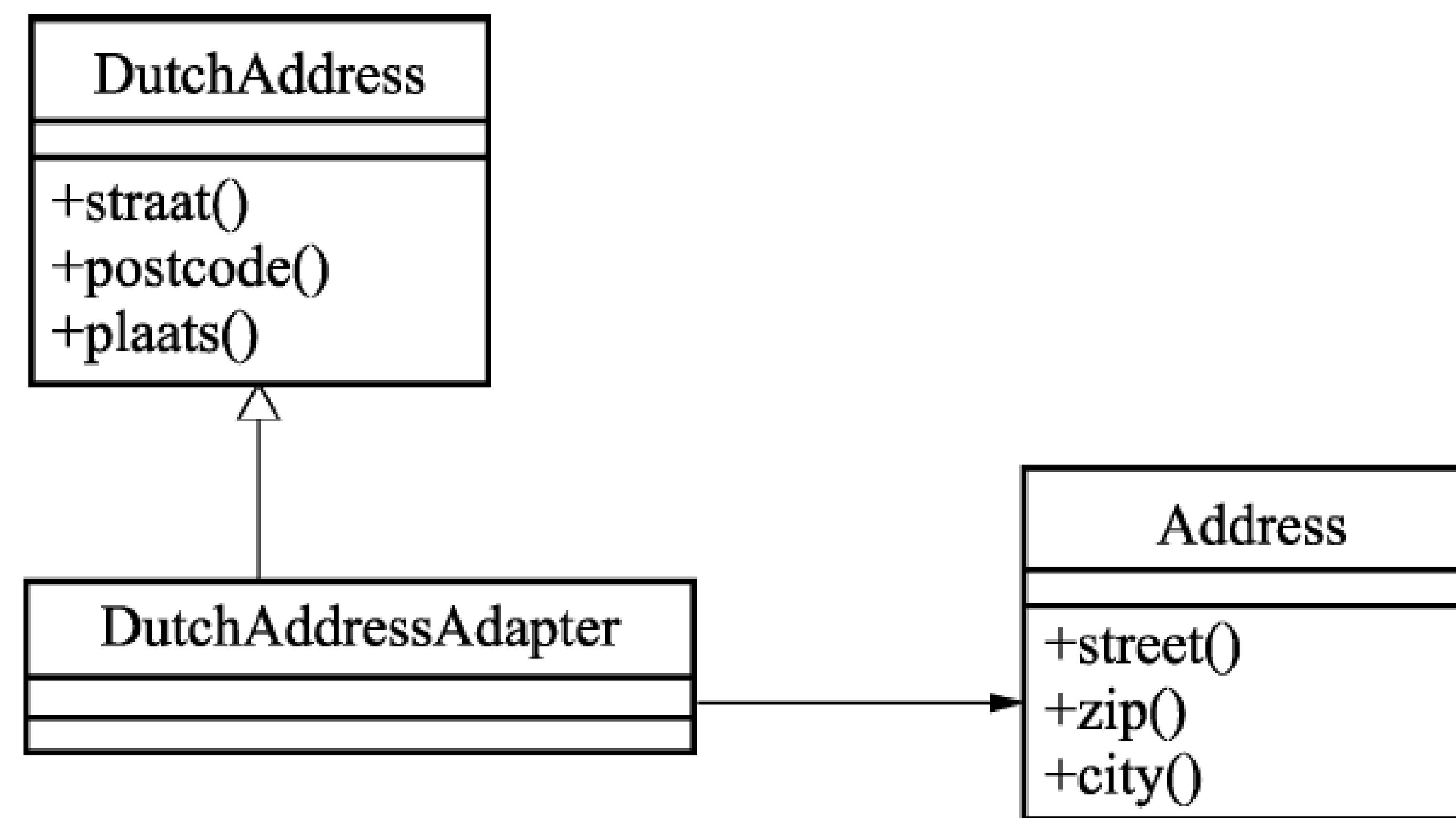


图 6-52　适配器模式类图

【Java 代码】

```
import java.util.*;
Class Address{
public void street() {//实现代码省略 }
public void zip(){//实现代码省略 }
public void city() {//实现代码省略 }
//其他成员省略
}
class DutchAddress{
public void straat(){//实现代码省略 }
public void postcode(){//实现代码省略 }
public void plaats(){//实现代码省略 }
//其他成员省略
}
class DutchAddressAdapter extends DutchAddress {
private (1);
public DutchAddressAdapter (Address addr){
address= addr;
}
public void straat() {
(2);
}
public void postcode() {
(3);
}
public void plaats(){
(4);
}
//其他成员省略
}
class Test {
public static void main(String[] args) {
Address addr=new Address();
(5);
System.out.println("\n The DutchAddress\n");
testDutch(addrAdapter);
```

```
}
Static void testDutch(DutchAddress addr){
addr.straat();
addr.postcode();
addr.plaats();
}
}
```

6.2.4 同步练习参考答案

1.

(1) abstract。 (2) null。 (3) List。 (4) childList。

(5) printTree(file)。

2.

(1) this.imp。 (2) ImageImp。 (3) imp.doPaint(m)。 (4) new BMP()。

(5) new WinImp()。 (6) image1.setImp(imageImp1)。 (7) 17。

3.

(1) Document。 (2) aDocument。 (3) !canOpenDocument(docName)。

(4) Document。 (5) doCreateDocument()。 (6) adoc.open(docName)。

(7) adoc.read(docName)。 (8) addDocument(adoc)。

4.

(1) commands[button]。 (2) turnLight(0)。 (3) setChannel(1)。

(4) lightCommand。 (5) RemoteController。 (6) Light。 (7) TV。

5.

(1) ProcessRequest(aRequest)。 (2) Approver。 (3) super。

(4) Tammy。 (5) Meeting。 (6) Sam。 (7) Larry。

6.

(1) Interface。 (2) Interface。 (3) flyBehavior。

(4) quackBehavior。 (5) flyBehavior.fly。 (6) quackBehavior.quack()。

(7) abstract。 (8) FlyNoWay()。 (9) Squeak()。

7.

(1) state==CLOSED||state==CLOSING。

(2) state==OPENING||state==STAYOPEN。

(3) state==OPEN。

(4) state.click()。

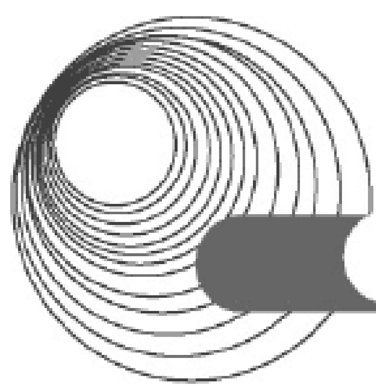
(5) state.timeout()。

(6) state.complete()。

(7) door.setState(door.OPENING)。

8.

(1) implements Iterable<Order>。 (2) orders.iterator()。



(3) catalog。　　(4) sales。　　(5) OrderItem item。

9.

(1) Address address。

(2) address.street()。

(3) address.zip()。

(4) address.city()。

(5) DutchAddress addrAdapter=new DutchAddressAdaptor(addr)。

6.3 本 章 小 结

本章知识点在 2009 年的新大纲中有些改动，原来面向对象的程序设计有对 Visual Basic 的要求，现在只需掌握 C++和 Java 中的一种就可以了。

根据近几年软件设计师水平考试试题分布情况来看，面向对象的程序设计的题目都是放在选做部分的，而且这两道题目用的题面是一样的，只是编程语言不同而已。这两道题目占的分数也是一定的，C++和 Java 的程序填空各占 15 分，总共 30 分。

C++和 Java 程序设计都是以填空的形式出题的，要求考生掌握这两种编程语言中任意一种的基本概念、基本语法和程序设计要注意的问题。在这个基础上再认真理解题目的条件和给出的那部分程序，就比较容易把程序补充完整了。



第 7 章　样 卷 模 拟

7.1　样　　卷

7.1.1　样卷一

全国计算机技术与软件专业技术资格(水平)考试样卷一

试题一~试题四是必答题

试题一(15 分)

阅读下列说明和数据流图，回答问题 1~问题 3。

【说明】

某考务处理系统的主要功能是考生管理和成绩管理。

(1) 对考生送来的报名表进行检查。

(2) 对合格的报名表编好准考证号码后将准考证送给考生，将汇总后的考生名单送给阅卷站。

(3) 对阅卷站送来的成绩表进行检查，并根据考试中心指定的合格标准审定合格者。

(4) 填写考生通知单(内容包含该考生的准考证号、姓名、各课程成绩及最终合格/不合格标志)，送给考生。

(5) 根据考生信息及考试成绩，按地区、年龄、文化程度和职业进行成绩分类统计及试题难度分析，产生统计分析表。

考务处理系统的顶层数据流图如图 7-1 所示；0 层数据流图如图 7-2 所示；加工 2 细化图如图 7-3 所示。

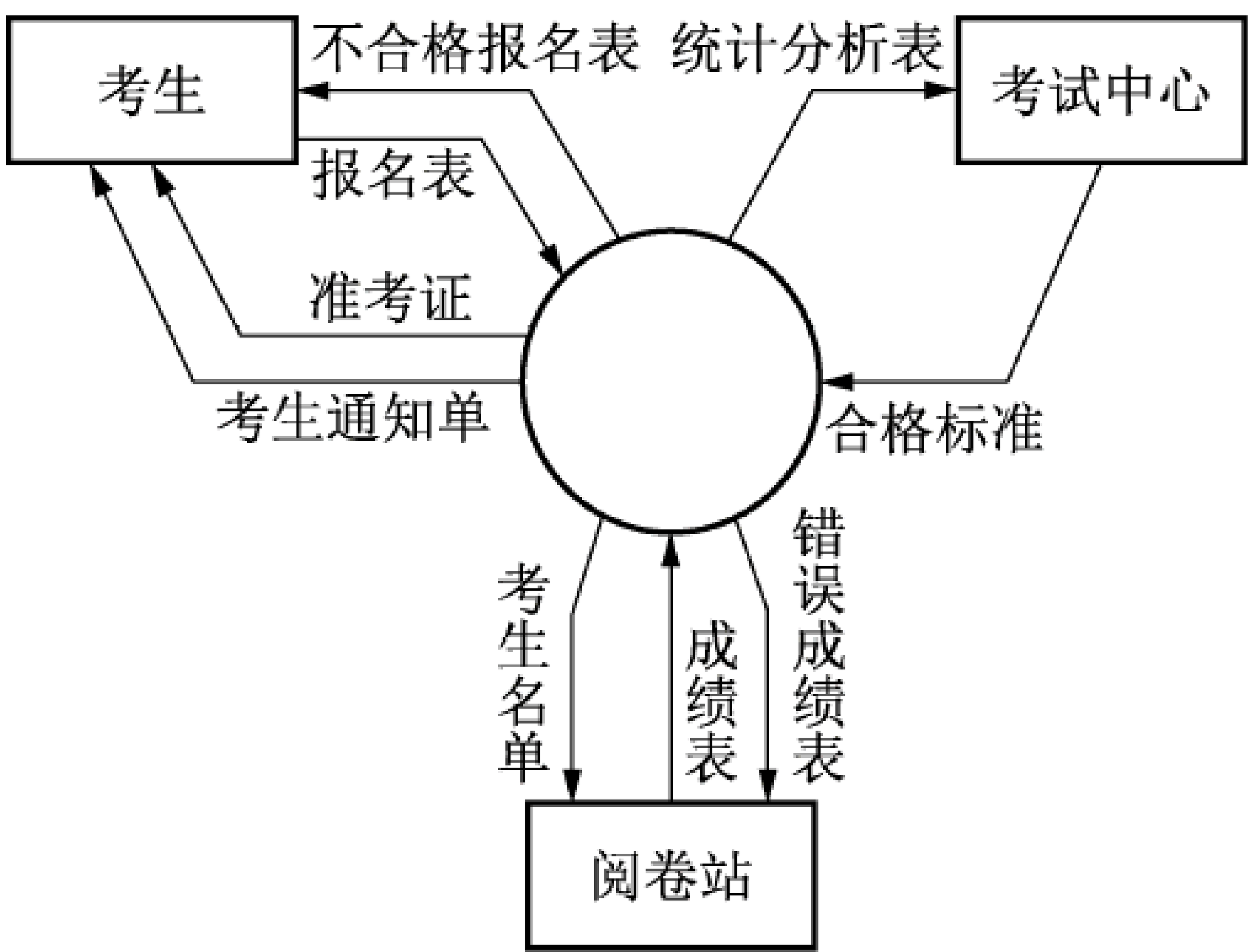


图 7-1　顶层数据流图

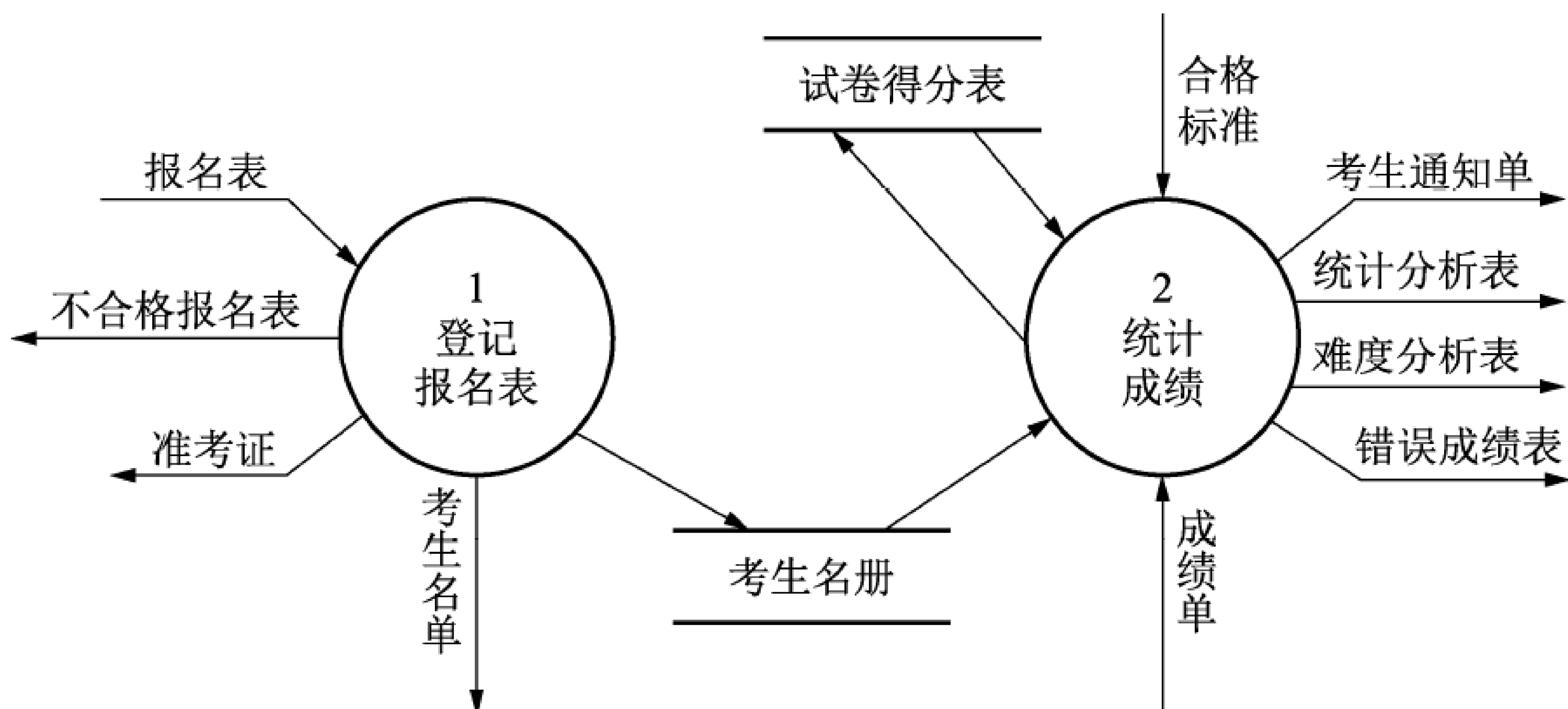


图 7-2　0 层数据流图

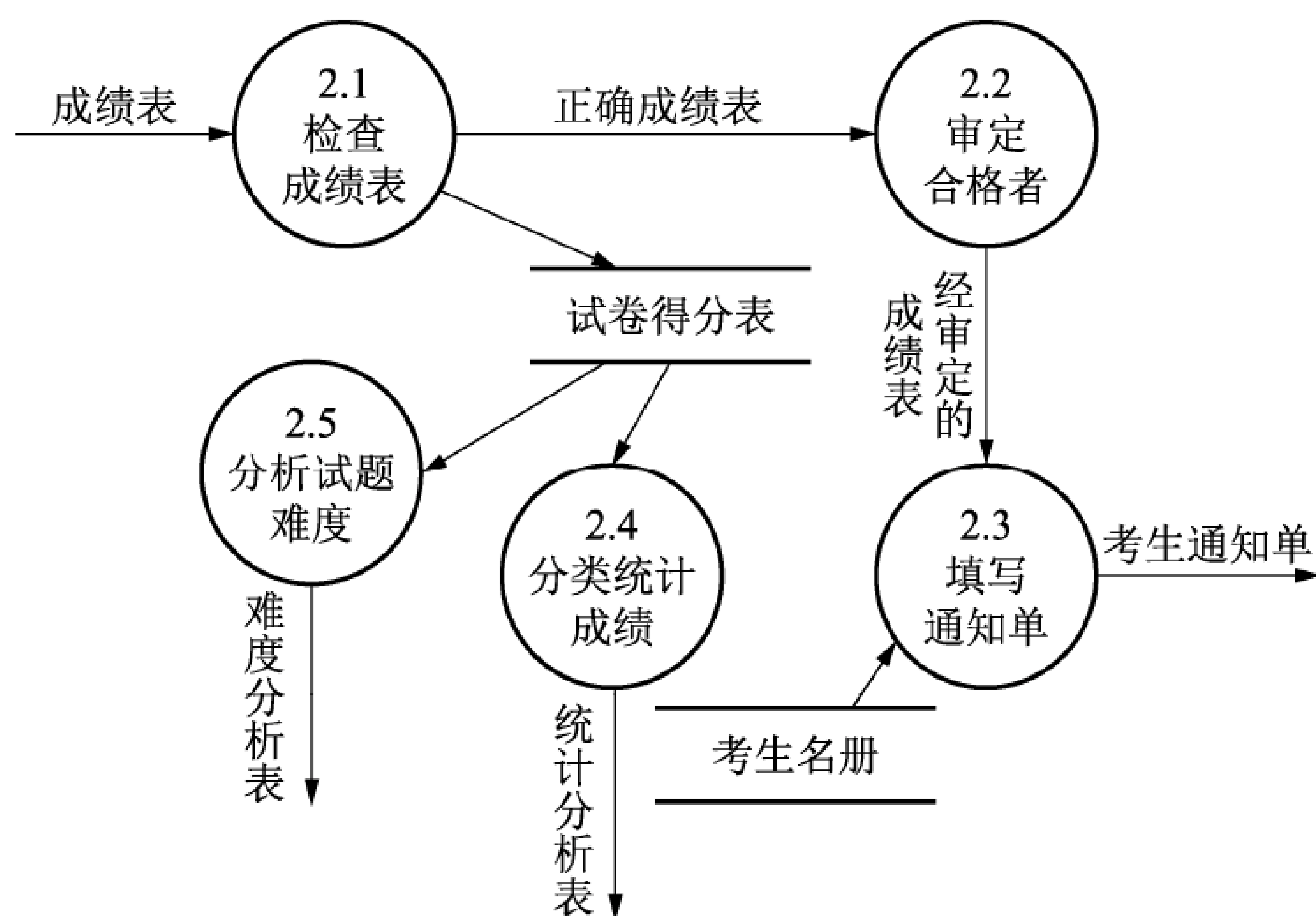


图 7-3　加工 2 细化图

【问题 1】(2 分)

指出哪张图的哪些文件可以不必画出。

【问题 2】(6 分)

数据流图 7-3 中缺少 3 条数据流，请直接在图中添加。

【问题 3】(7 分)

根据系统功能和数据流图填充下列数据字典条目中的空(1)处和空(2)处。

试题得分表=准考证号+｛课程名+成绩｝；

考生名册=报名号+准考证号+姓名+通信地址+出生年份+文化程度+职业；

考生通知单=__(1)__。

报名表=__(2)__。

试题二(15 分)

阅读下列说明和 E-R 图，回答问题 1~问题 3。

【说明】

设有关于银行借贷管理系统的 E-R 图如图 7-4 所示。图中矩形表示实体，圆表示属性，双圆表示关键字属性，菱形表示实体间的联系。为了答题的方便，图中的实体和属性同时给出了中英文两种名字，回答问题时只需写出英文名即可。

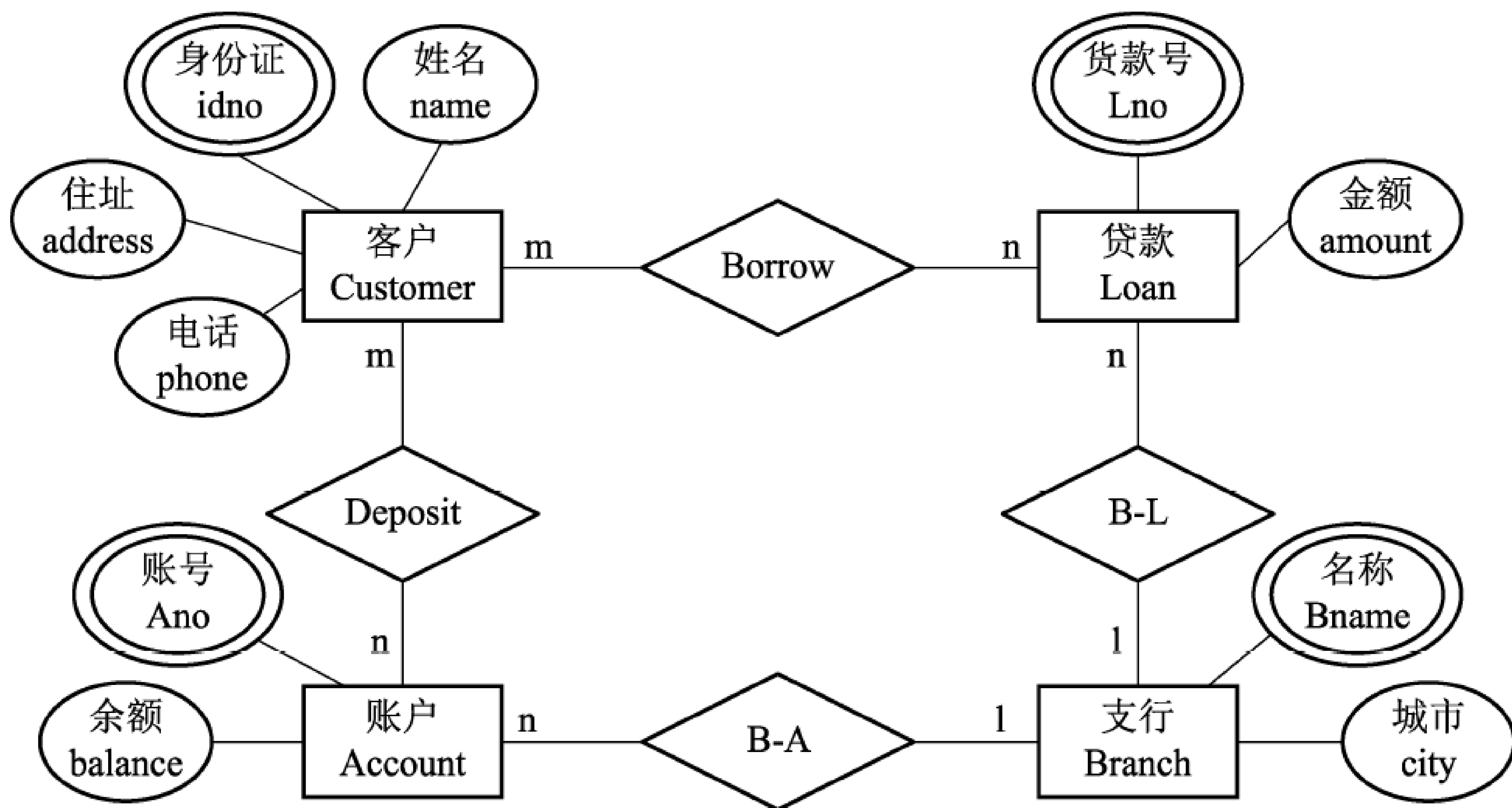


图 7-4　银行借贷管理系统 E-R 图

【问题 1】(10 分)

根据 E-R 图中给出的词汇，按照“有关模式名(属性 1,属性 2,……)”的格式，将此 E-R 图转换为关系模式，并指出每个关系模式中的主键和外键，其中模式名根据需要取实体名或联系名。要求其中的关系模式至少属于第三范式。

【问题 2】(2 分)

以下的 SQL 语句是用于查询“在该银行中一笔贷款贷给多个(至少两个)客户的所有贷款号和发放贷款的支行名称”的不完整语句，请在空缺处填入正确的内容。

```
SELECT Borrow.Lno, Bname
FROM Borrow, Loan
WHERE   (1)
GROUP BY Borrow.Lno
HAVING   (2)  ;
```

【问题 3】(3 分)

假设这家银行有若干个节点，每个节点运行一个数据库系统。假设这些节点之间唯一的交互是用电子方式相互传送款项，这样的系统是分布式数据库系统吗？为什么？

试题三(15 分)

阅读下列说明及图示，回答问题 1~问题 3。

【说明】

某图书馆信息管理系统的主要功能如下。

图书馆雇有若干管理员，各自具有编码、姓名等属性。管理员可上岗，也可下岗。

图书馆中备有若干图书，每本图书有书号、书名、出版社、价格等属性。图书馆不定期地购买并注册新图书供读者借阅，也可将报废的旧书注销以停止借阅。

图书馆可为众多读者提供服务。每位读者在借阅之前需注册姓名、性别、地址、E-mail等内容。读者可在终端上查询。每位读者最多可同时借 5 本书。每本图书借期 30 天；若有一本书超期，则不可再借其他图书。一本书超期一天罚款 0.1 元。若一本书超期 3 个月不归还，则发布通告。若读者的图书丢失，在罚款处理之前不能借书，每本报失的图书罚款该书价的两倍。注册新读者不受限制；而注销读者之前，该读者必须归还所有借阅的图书，或者报失并接受罚款。

UML 的用例图及借书和还书的协作图如图 7-5 和图 7-6 所示。

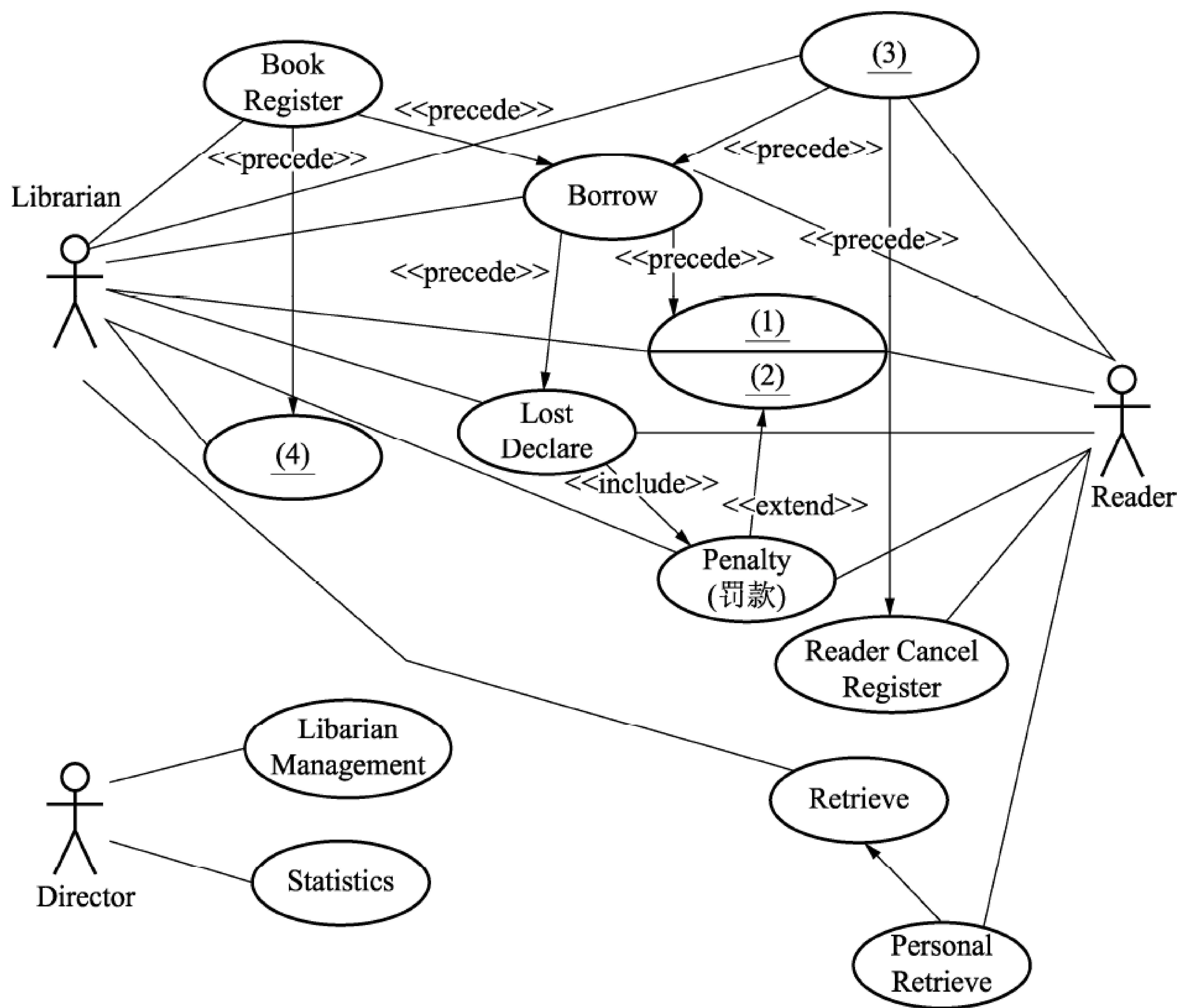


图 7-5　UML 的用例图

【问题 1】(6 分)

在需求分析阶段，采用 UML 的用例图描述系统功能需求，如图 7-5 所示。指出图 7-5 中(1)~(4)处分别是哪个用例。

【问题 2】(5 分)

图 7-6 采用协作图描述借书和还书两个动态过程的交互关系。在 UML 中，重复度(Multiplicity)定义了某个实体的一个实例可以与另一个类的多个实例相关联。指出图 7-6 中空(5)处、空(6)处的重复度分别是多少。

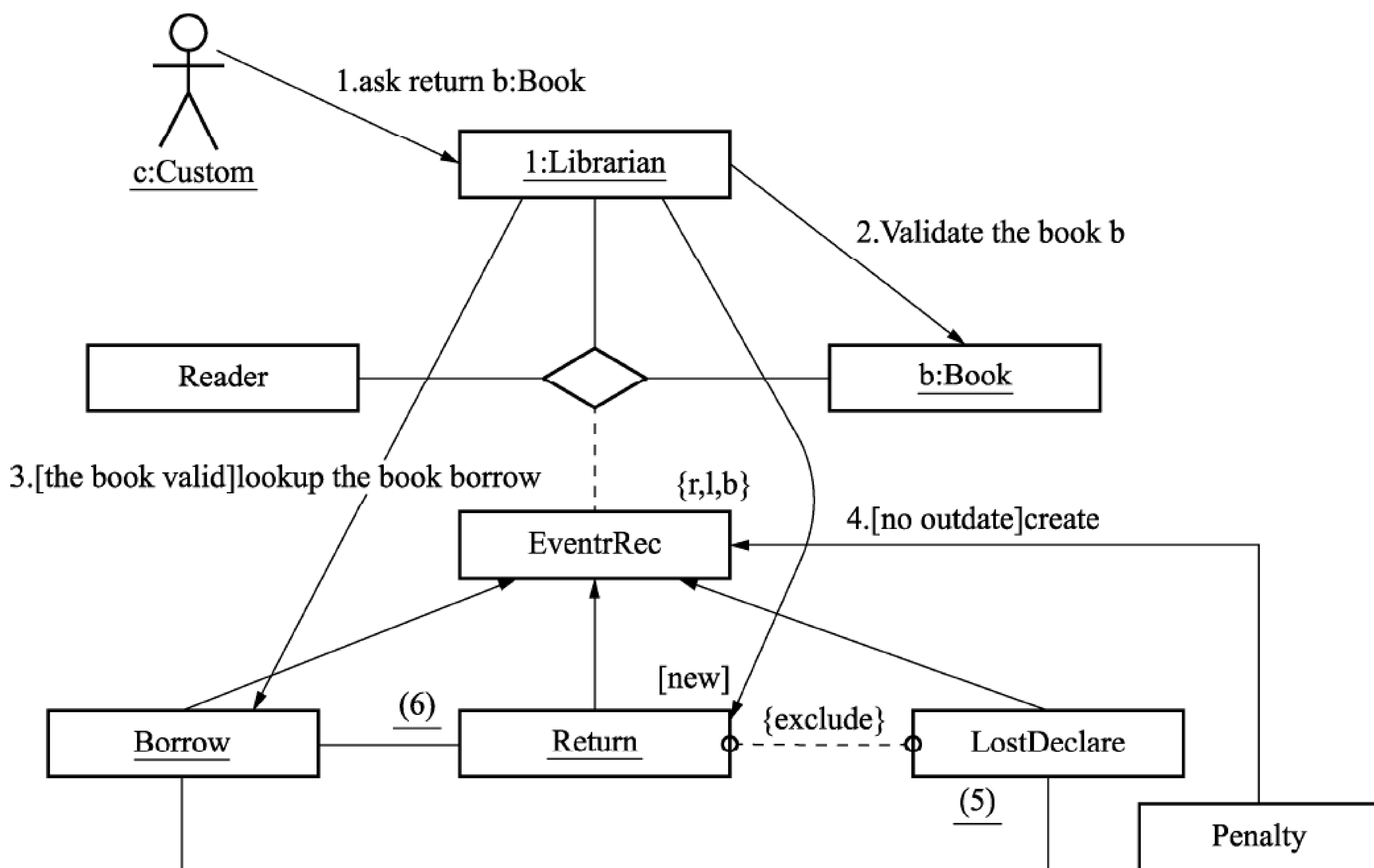


图 7-6　借书和还书的协作图

【问题 3】(4 分)

指出 UML 中全局、局部、参数、自我、投票、广播、创建、注销和临时 9 个约束对于链接角色、消息和对象的作用。

试题四(15 分，每空 3 分)

阅读下列说明、流程图和算法，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

如图 7-7(a)所示的流程图用 N-S 盒图形式描述了数组 A 中的元素被划分的过程。其划分方法是：以数组中的第一个元素作为基准数，将小于基准数的元素向低下标端移动，而大于基准数的元素向高下标端移动。当划分结束时，基准数定位于 A[i]，并且数组中下标小于 i 的元素的值均小于基准数，下标大于 i 的元素的值均大于基准数。设数组 A 的下界为 low，上界为 high，数组中的元素互不相同。例如，对数组(4,2,8,3,6)，以 4 为基准数的划分过程如图 7-7(b)所示。

【算法说明】

将上述划分的思想进一步用于被划分出的数组的两部分，就可以对整个数组实现递增排序。设函数 int p(int A[], int low, int high)实现了上述流程图的划分过程，并返回基准数在数组 A 中的下标。递归函数 void sort(int A[], int L, int H)的功能是实现数组 A 中元素的递增排序。

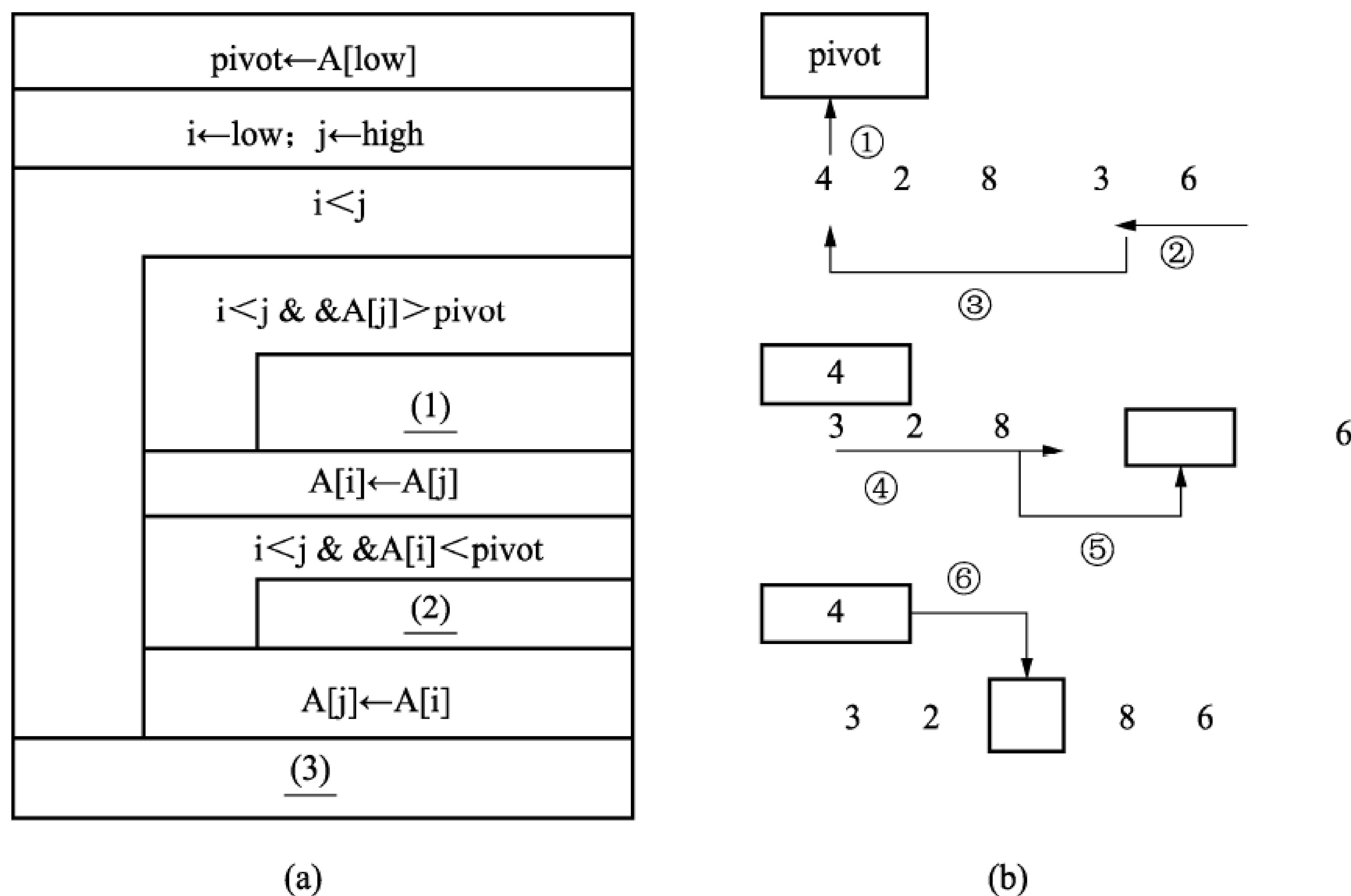


图 7-7　N-S 流程图

【算法】

```
void sort(int A[],iht L,int H) {
  if ( L < H ) {
    k=p(A,L,R);           //p()返回基准数在数组 A 中的下标
    sort( (4) );           //小于基准数的元素排序
    sort( (5) );           //大于基准数的元素排序
    }
}
```

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(15 分，每空 1.5 分)

阅读下列说明和程序，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

定义一个多边形结构 struct polygon，实现以下功能：①建立该结构的链表，create 函数是创建链表，每输入一个节点的数据，就把该节点加入到链表当中，它返回创建的链表的头指针；②显示链表的各个节点数据，节点数据包括多边形顶点数、各顶点的纵横坐标，当多边形顶点数为 0 时，链表创建结束；③编写一个函数 disp，删除链表中的所有节点。需要注意的是：要先释放节点数据内存单元，再删除节点，如果在释放节点数据内存单元之前删除节点，则无法找到节点数据内存单元的地址，也就无法释放数据的内存单元。

【程序】

```
#include "iostream.h"
#include "iomanip.h"
struct polygon
```

```
{
  int n;
  int *x;
  int *y;
  polygon *next;
};
void Push(polygon*& head, int n)
{
  polygon* newNode = new polygon;
  newNode = new polygon;
  newNode->next=__(1)__;
  newNode->x = new int[n];
  newNode->y = new int[n];
  newNode->n=__(2)__;
  for(int i=0; i<=__(3)__; i++){
    cout<<"请输入多边形各顶点 x、y 坐标，坐标值之间用空格分隔：";
    cin>>newNode->x[i]>>newNode->y[i];
  }
  __(4)__= head; // 在 head 前不需要额外的*
  head = newNode;
}
polygon *create()
{
  polygon* head = NULL;
  polygon* tail;
  int n;
  cout<<"请输入多边形顶点的个数(顶点个数为 0 时结束)：";
  cin>>n;
  if(n==0) return __(5)__;
  Push(head,__(6)__);
  tail = head;
  cout<<"请输入多边形顶点的个数(顶点个数为 0 时结束)：";
  cin>>n;
  while(n!=0)
  {
  Push(tail->next,__(7)__); // 在 tail->next 增加节点
    tail = tail->next; // tail 指向下一个节点
    cout<<"请输入多边形顶点的个数(顶点个数为 0 时结束)：";
    cin>>n;
  }
  return head;
}
void disp(polygon *head)
{
  int i, No=1;
  cout<<setw(10)<<"x"<<setw(6)<<"y"<<endl;
  while(head!=NULL)
  {
  cout<<"第"<<No<<"节点："<<endl;
```

```
    for(i=0;i<=head->n-1;i++)
    cout<<setw(10)<<head->x[i]<<setw(6)<<head->y[i]<<endl;
    ___(8)___;
    head=___(9)___;
  }//通过 while 循环显示节点信息
}
void del(polygon *head)
{
  polygon *p;
  while(head!=NULL)
  {
    p=___(10)___;
head=head->next;
    delete p->x;
delete p->y;
    delete p;
    }//通过 while 循环删除节点信息
}
void main()
{
polygon *head;
  head=create();
  disp(head);
  del(head);
}
```

试题六(共 15 分)

阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

对多个元素的聚合进行遍历访问时，需要依次推移元素。例如，对数组通过递增下标的方式来进行遍历，数组下标功能抽象化、一般化的结果称为迭代器(Iterator)。以下程序模拟将书籍(Book)放到书架(BookShelf)上并依次输出书名。这样就要涉及遍历整个书架的过程。该过程可使用迭代器 Iterator 实现。图 7-8 显示了各个类间的关系。以下是 C++语言实现，能够正确编译通过。

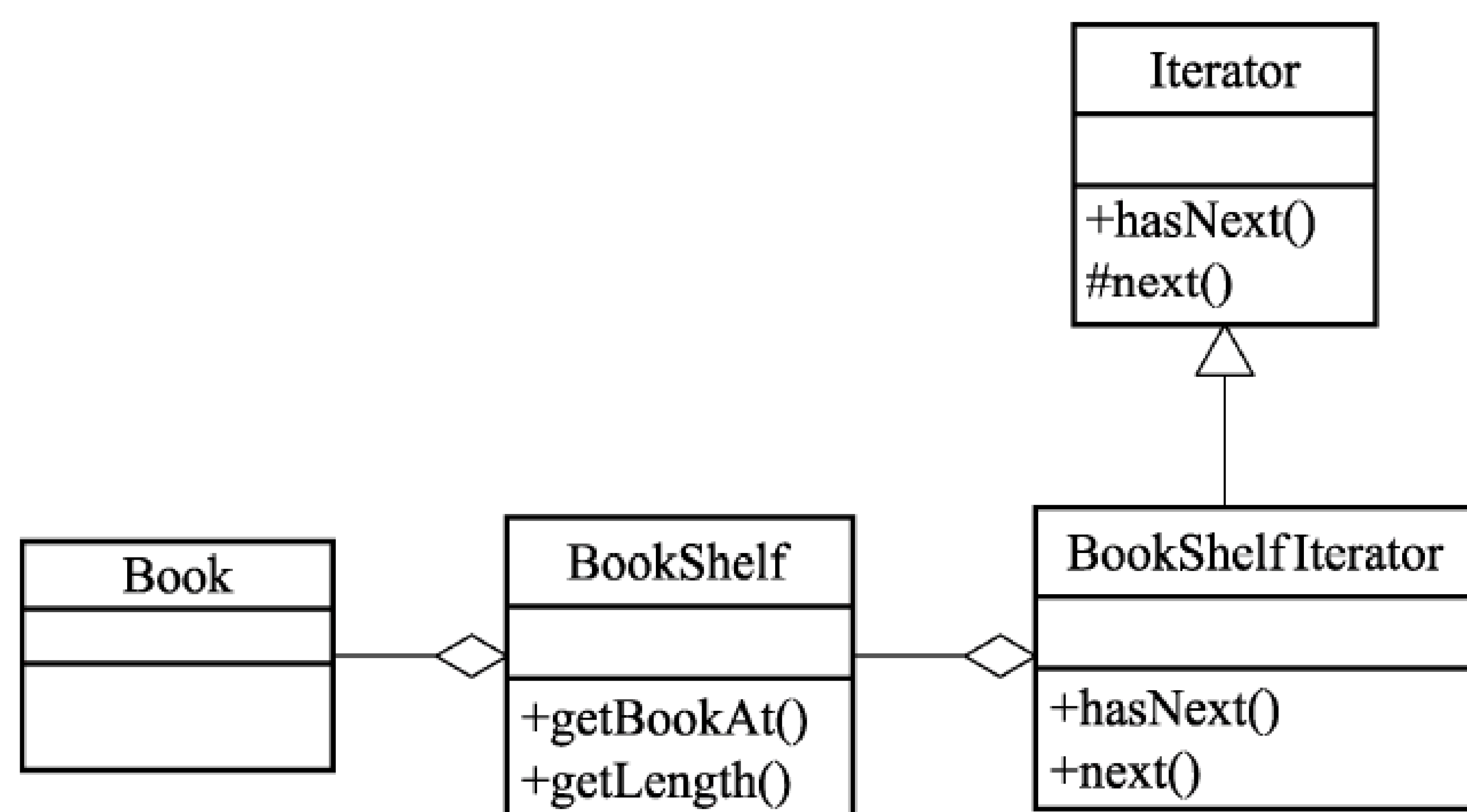


图 7-8 类图关系

【C++程序】

```
template<class  (1) >
class Iterator{
public:
    virtual bool hasNext() = 0;
     (2)  Object* next() = 0;
};

class Book{
//省略具体方法和属性
};

class BookShelf{
private:
    vector<Book> books;
public:
    BookShelf(){
    }
    Book* getBookAt(int index){
        return &books[index];
    }
    int getLength(){
        return books.size();
    }
};

template<class Object>
class BookShelfIterator:public  (3) {
private:
    BookShelf *bookShelf;
    int index;
public:
    BookShelfIterator(BookShelf *bookShelf){
        this->bookShelf = bookShelf;
        index = 0;
    }
    bool hasNext(){//判断是否还有下一个元素
        if(index < bookShelf->getLength()){
            return true;
        }else{
            return false;
        }
    }
    Object* next(){//取得下一个元素
        return bookShelf->getBookAt(index++);
    }
};
```

```
int main()
{
    BookShelf bookShelf;
    //将书籍上架，省略代码
    Book *book;
    Iterator<Book> *it = new BookShelfIterator<Book>( (4) );
    while( (5) ){//遍历书架，输出书名
        book = (Book*)it->next();
        /*访问元素*/
    }
    return 0;
}
```

试题七(共 15 分)

阅读以下说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

对多个元素的聚合进行遍历访问时，需要依次推移元素。例如，对数组通过递增下标的方式来进行遍历，数组下标功能抽象化、一般化的结果称为迭代器(Iterator)。以下程序模拟将书籍(Book)放到书架(BookShelf)上并依次输出书名。这样就要涉及遍历整个书架的过程。该过程可使用迭代器 Iterator 实现。图 7-9 显示了各个类间的关系。以下是 Java 语言实现，能够正确编译通过。

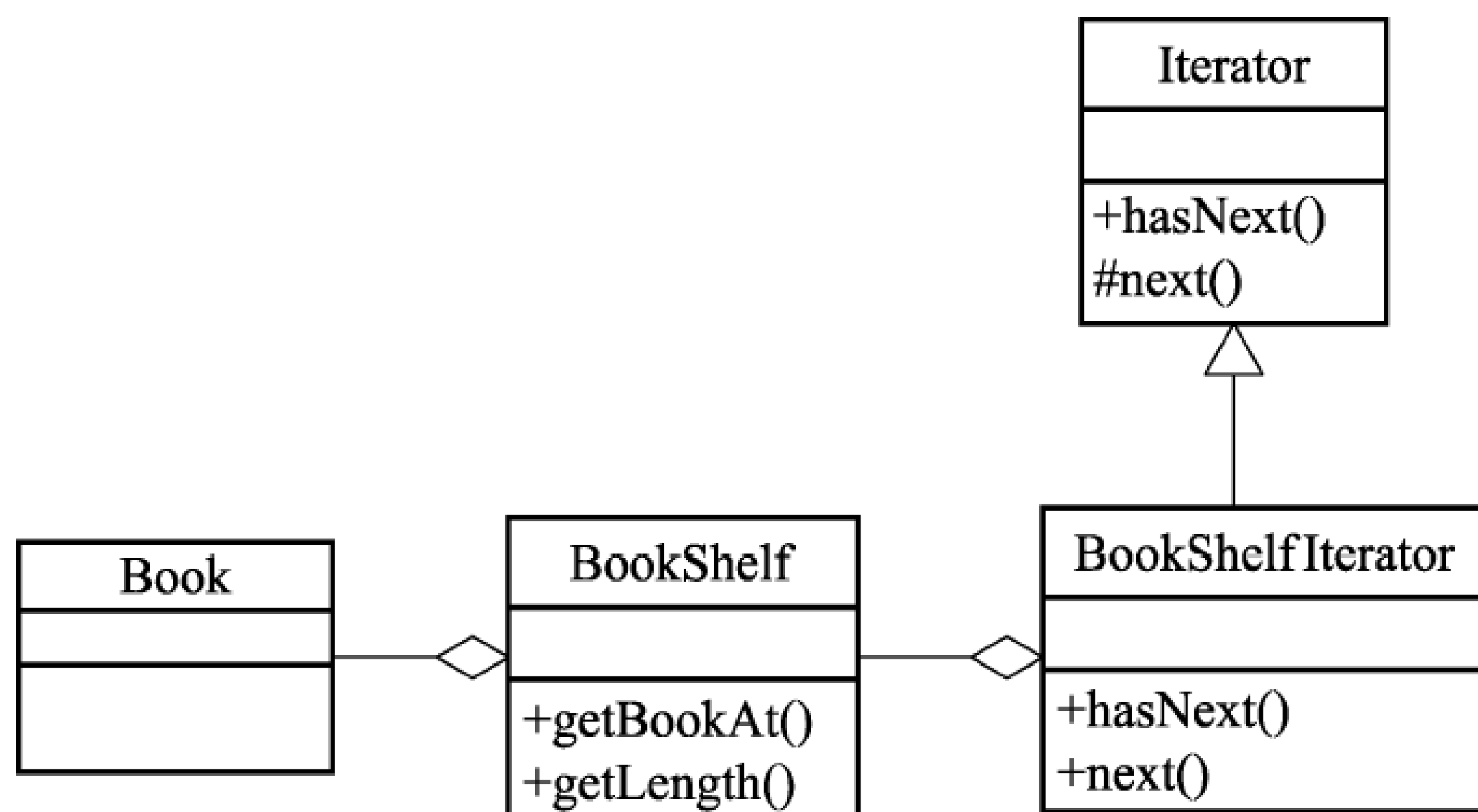


图 7-9 类图关系

【Java 程序】

```
//Iterator.java 文件
public interface Iterator {
    public abstract boolean hasNext();
    public abstract Object next();
}

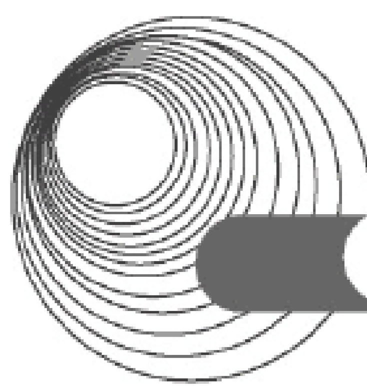
//Aggregate.java 文件
public interface Aggregate {
    public abstract Iterator iterator();
}
```

```
//Book.java
public class Book {
    //省略具体方法和属性
}

//BookShelfIterator.java 文件
public class BookShelfIterator  (1)  Iterator{
    private BookShelf bookShelf;
    private int index;
    public BookShelfIterator(BookShelf bookShelf){
        this.bookShelf = bookShelf;
        this.index = 0;
    }
    public boolean hasNext(){//判断是否还有下一个元素
        if(index < bookShelf.getLength()){
            return true;
        }else{
            return false;
        }
    }
    public Object next(){//取得下一个元素
        Book book = bookShelf.getBookAt(index);
        index++;
        return book;
    }
}

//BookShelf.java
import java.util.Vector;
public class BookShelf {
    private Vector books;
    public BookShelf(int initialsize){
        this.books = new Vector(initialsize);
    }
    public Book getBookAt(int index){
        return (Book)books.get(index);
    }
    public int getLength(){
        return books.size();
    }
    public Iterator iterator(){
        return new BookShelfIterator( (2) );
    }
}

//Main.java 文件
public class Main {
    public static void main(String[] args){
        BookShelf bookShelf = new BookShelf(4);
```




```
            //将书籍上架，省略代码
            Iterator it = bookShelf.(3);
            while((4)){//遍历书架，输出书名
                Book book = (Book)it.(5);
                System.out.println(""+book.getName());
            }
        }
    }
```

7.1.2 样卷二

全国计算机技术与软件专业技术资格(水平)考试样卷二

试题一~试题四是必答题

试题一(15 分)

阅读下列说明和有关的图，回答问题 1~问题 4，将解答填入答题纸的对应栏内。

【说明】

某制造企业的物料出入库管理的工作流程分别叙述如下。

1. 出库工作流程

(1) 领料人提交领料单(每一种物料有一张领料单)。

(2) 仓库保管员根据领料计划单检验该领料单是否有效。

(3) 若经检验没有相应的领料计划，则通知领料人该领料单无效。

(4) 若领料单有效，仓库保管员根据领料单上的物料代码核对是否有足够的库存。

(5) 若没有足够的库存，仓库保管员向领料人发缺货单。

(6) 若有足够的库存，仓库保管员在领料单上签字，并登记出库单，修改物料主文件中的现有库存数；相应的物料出库，物料清单交领料人。

2. 入库工作流程

(1) 采购员提交入库申请单(每一种物料有一张入库申请单)。

(2) 仓库保管员根据采购计划单验收入库申请单。

(3) 若验收发现没有相应的采购计划，则仓库保管员向采购员发无效申请单。

(4) 若验收合格，则仓库保管员向检验员申请物料检验；检验员根据检验结果填写物料检验单。

(5) 如果物料或供货方不合格，则向采购员发出退货单。

(6) 如果检验合格，则仓库保管员登记入库单，修改物料主文件中的现有库存数，相应的物料入库。

为便于及时了解库存情况、核查出入库情况，该企业决定将上述人工流程由计算机来实现。在设计该系统时，采用了两种方法：结构化方法和面向对象方法。

图 7-10 给出了物料出入库系统的数据流图，图中的数据流并没有画全，需要考生填补。图 7-11 给出了采用面向对象方法所认定出的类。

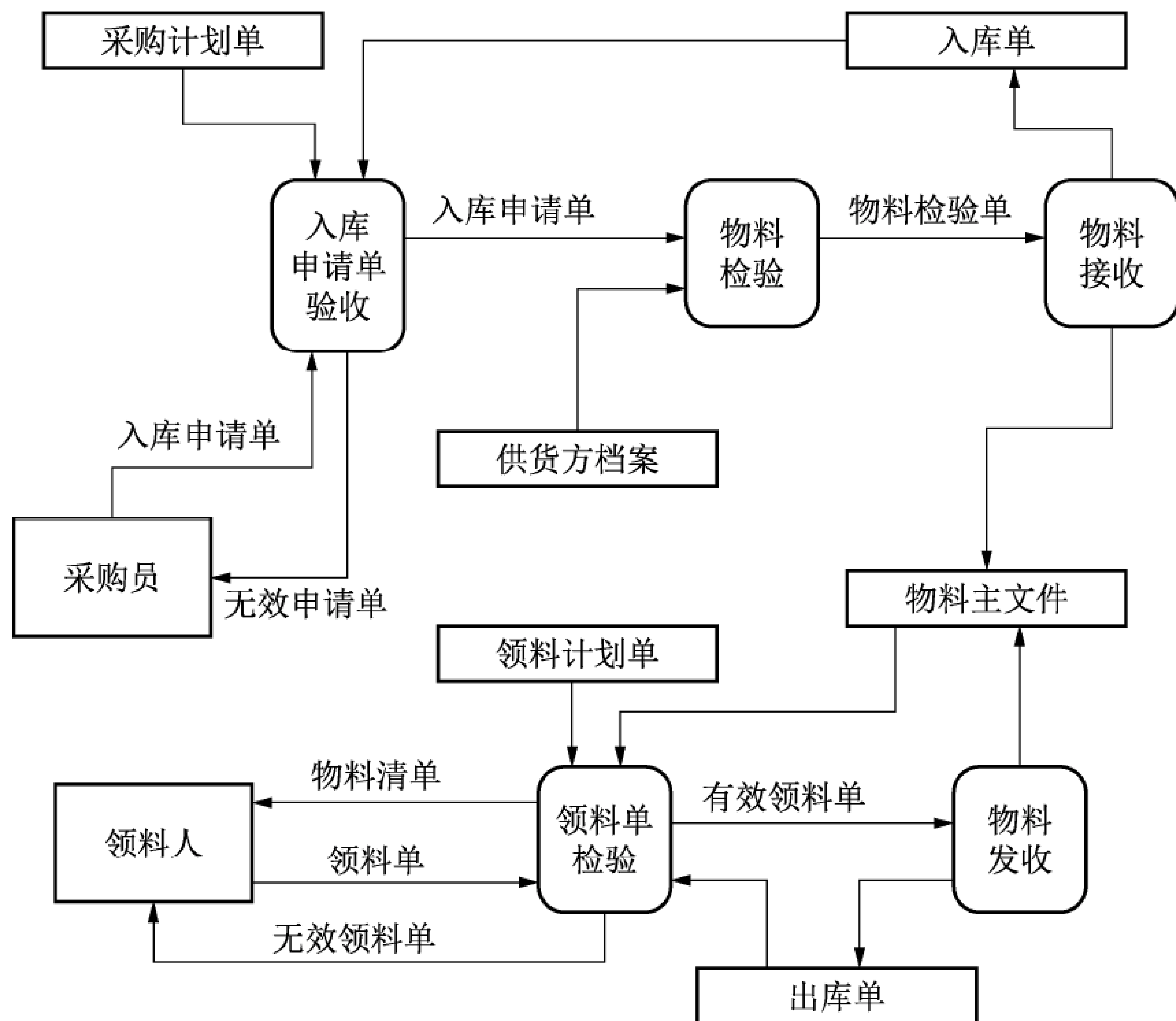


图 7-10　物料出入库系统的数据流图

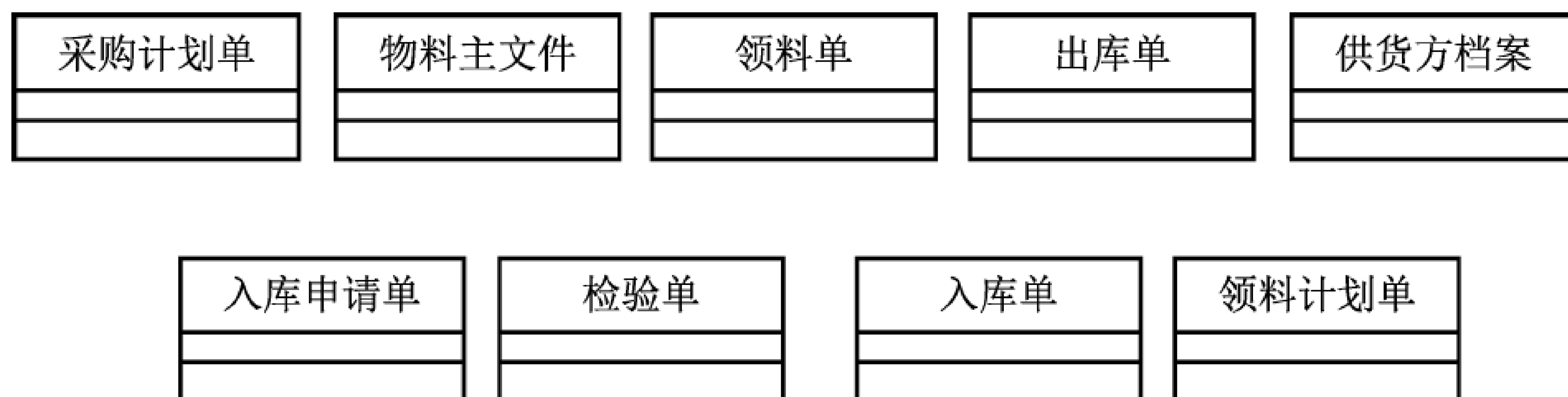


图 7-11　物料出入库系统中的类

【问题 1】

图 7-10 中缺少了哪些数据流？请指明每条数据流的名称、起点和终点。

【问题 2】

给出“领料单”和“入库申请单”这两个类至少应具有的属性。

【问题 3】

为建立功能完善的库存管理系统，除了查询、统计、报表输出功能外，还应具有哪些对提高企业效益至关重要的功能？

【问题 4】

用面向对象方法设计的类中，有一些类的对象是需要持久存储的，这样的类一般需要映射到关系数据库模式中。请指出图 7-11 中哪些类需要做这样的映射。

试题二(15 分)

阅读下列说明，回答问题 1~问题 4，将解答填入答题纸的对应栏内。

【说明】

某超市的销售业务由一个销售业务管理系统进行管理，该系统每完成一次交易都需要提供顾客发票，其格式如表 7-1 所示。

表 7-1　超市购物发票表

		××××超市购物发票				
	顾客代码:			发票号码:		
	交易日期:			收银员代码:		
	商品代码	商品名称	单　价	数　量	金　额	
			总计:	件	元	

对于这样一个销售业务管理系统，分别给出了以下两种关系数据库的设计(下画线表示主关键字)。

- 设计一：
  顾客 Customer(顾客代码 Cno,姓名 name,住址 address,联系电话 phone)
  收银员 Salesman(收银员代码 Sno,身份证号 idno,姓名 name,住址 address,联系电话 phone)
  商品 Merchandise(商品代码 Mno,商品名称 Mname,价格 price)
  发票 Invoice(发票号码 Ino,交易日期 Idate,顾客代码 Cno,收银员代码 Sno,商品代码 Mno,单价 unitprice,数量 amount)
- 设计二：
  顾客 Customer(顾客代码 Cno,姓名 name,住址 address,联系电话 phone)
  收银员 Salesman(收银员代码 Sno,身份证号 idno,姓名 name,住址 address,联系电话 phone)
  商品 Merchandise(商品代码 Mno,商品名称 Mname,价格 price)
  发票 Invoice(发票号码 Ino,交易日期 Idate,顾客代码 Cno,收银员代码 Sno)
  发票明细 Invoicedetail(发票号码 Ino,商品代码 Mno,单价 unitprice,数量 amount)

【问题 1】(4 分)

设计一中的关系模式 Invoice 最高满足第几范式？为什么？设计一和设计二哪个更加合理？为什么？

【问题 2】(5 分)

根据设计二中的关系模式，以下 SQL 语句是用于“建立 2005 年 1 月期间每张发票的发票号、交易日期、交易商品件数和交易总金额的视图”的不完整语句，请填补其中的空缺。

```
CREATE VIEW Invoice_total ___(1)___
SELECT Invoice.Ino,Idate,___(2)___,___(3)___
FROM Invoice,Invoicedetail
WHERE ___(4)___ AND
      Idate BETWEEN '2005-01-01' AND '2005-01-31'
GROUP BY ___(5)___ ;
```

【问题 3】(3 分)

根据设计二中的关系模式，以下 SQL 语句是用于“查询从未售出的商品信息”的不完整语句，请填补其中的空缺。

```
SELECT Mno,Mname,price
FROM Merchandise ___(1)___
WHERE ___(2)___
      (SELECT ___(3)___
       FROM Invoicedetail
       WHERE A.Mno= Invoicedetail.Mno);
```

【问题 4】(3 分)

设计二中的关系模式 Merchandise 由属性 price 表示商品价格，关系模式 Invoicedetail 中的属性 unitprice 也表示商品价格。两个是否有必要同时存在？为什么？

试题三(15 分，每空 3 分)

阅读下列说明及图示，回答问题 1~问题 3。

【说明】

下面是某租车信息管理系统的介绍：该车库中备有若干车辆，每辆车有车号、车牌、车名、价格等属性。车库不定期地购买并注册新车供用户借用，也可将报废的旧车注销以停止租用。

车库可为众多用户提供服务。每个用户在借车之前需注册姓名、地址等内容。每个用户最多可同时借 3 辆车。每辆车借期 7 天；若有一辆车超期，则不可再借其他车。一辆车超期一天罚款 250 元。若一辆车超期 3 周不归还，则发布通告。若用户借的车丢失，在罚款处理之前不能借车，每辆报失的车罚款该车目前市价(包括折旧)的 1.2 倍。注册新用户不受限制；而注销用户之前，该用户必须归还所有借的车，或者报失并接受罚款。

【问题 1】(4 分)

分析车辆的状态和事件，指出图 7-12 中的(1)~(4)处分别是什么？

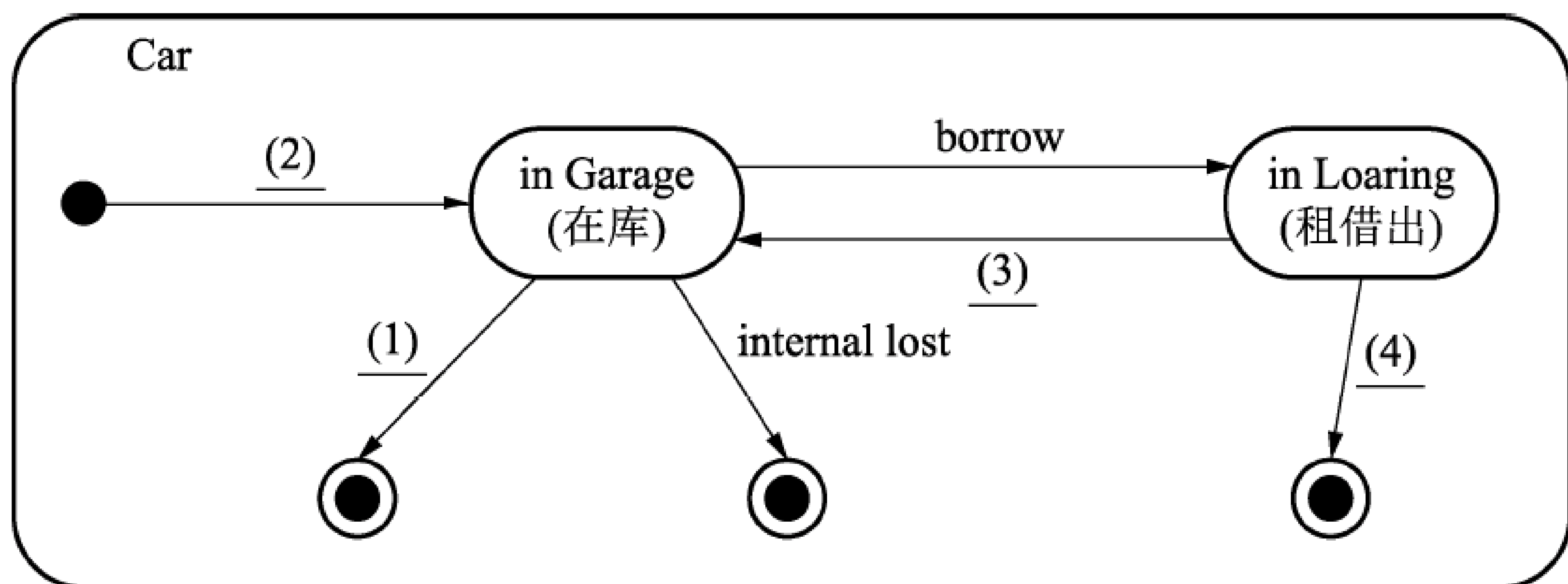


图 7-12　车辆的状态图

【问题 2】(6 分)

分析用户的状态和事件，指出图 7-13 中的(5)~(8)处分别是什么？(注意：用户与车辆在状态图中的关系。)

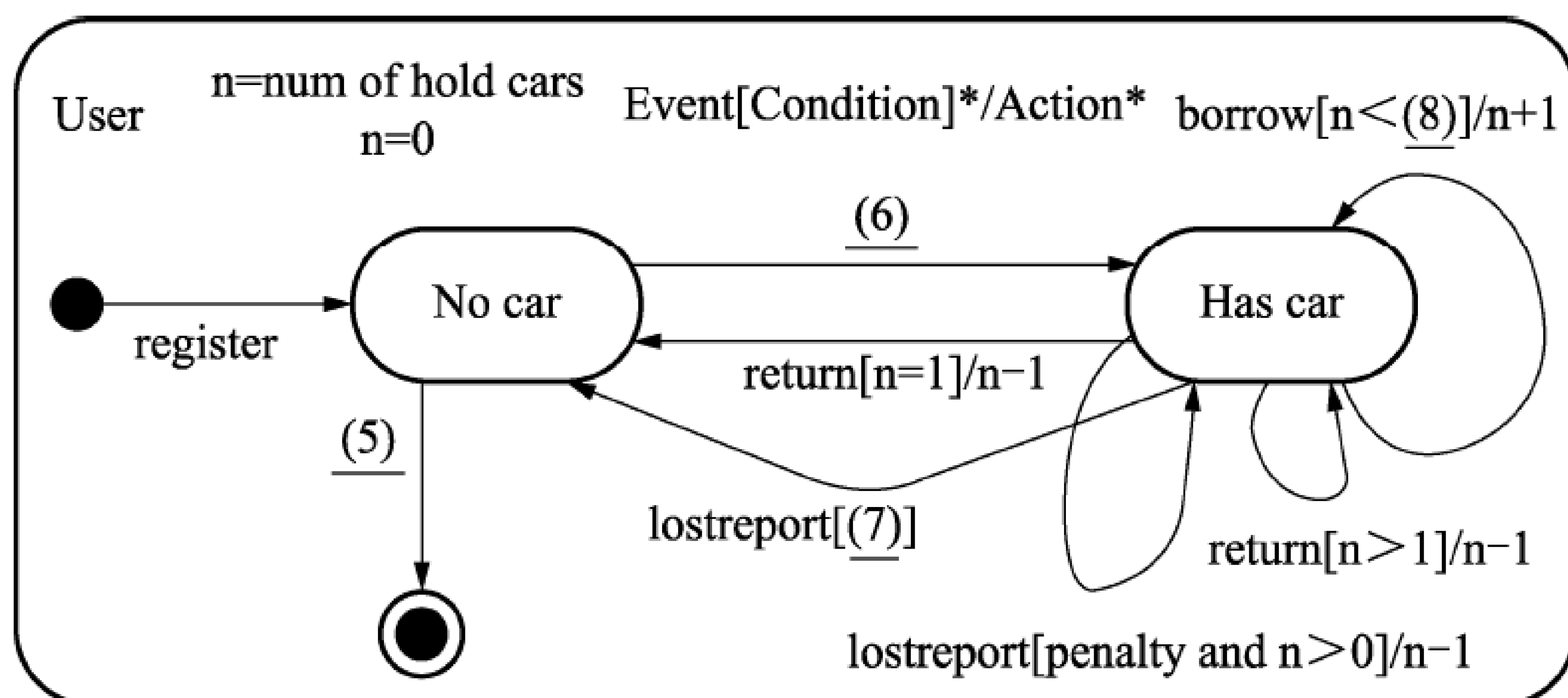


图 7-13　用户的状态图

【问题 3】(5 分)

指出 UML 中活动图的含义，并说明活动图和状态图的区别与联系。

试题四(15 分，每空 3 分)

阅读下列算法说明和算法，将应填入(n)的子句写在答题纸的对应栏内。

【说明】

下列最短路径算法的具体流程如下：首先构造一个只含 n 个顶点的森林，然后依权值从小到大从连通网中选择不使森林中产生回路的边加入到森林中去，直至该森林变成一棵树为止，这棵树便是连通网的最小生成树。该算法的基本思想是：为使生成树上总的权值之和达到最小，则应使每一条边上的权值尽可能地小，自然应从权值最小的边选起，直至选出 n-1 条互不构成回路的权值最小边为止。

【算法】

```
/*对图定义一种新的表示方法，以一维数组存放图中所有边，并在构建图的存储结构时将它构造为
一个“有序表”。以顺序表 MSTree 返回生成树上各条边*/

typedef struct {
    VertexType vex1;
    VertexType vex2;
    VRType weight;
}EdgeType;
typedef ElemType EdgeType;
typedef struct {                            // 有向网的定义
    VertexType vexs[MAX_VERTEX_NUM];        // 顶点信息
    EdgeType edge[MAX_EDGE_NUM];            // 边的信息
    int vexnum,arcnum;                      // 图中顶点的数目和边的数目
}ELGraph;
void MiniSpanTree_Kruskal(ELGraph G, SqList& MSTree){
// G.edge 中依权值从小到大存放有向网中各边
// 生成树的边存放在顺序表 MSTree 中
    MFSet F;
    InitSet(F, G.vexnum);                   // 将森林 F 初始化为 n 棵树的集合
```

```
    InitList(MSTree, G.vexnum);          // 初始化生成树为空树
    i=0; k=1;
while( k< __(1)__ ) {
e = G.edge[i];                            // 取第 i 条权值最小的边
/*函数 fix_mfset 返回边的顶点所在树的树根代号，如果边的两个顶点所在树的树根相同，则说
明它们已落在同一棵树上*/
    r1 = fix_mfset(F, LocateVex(e.vex1));
    r2 =  __(2)__  //返回两个顶点所在树的树根
    if (r1 __(3)__ r2) {                  // 选定生成树上第 k 条边
    if (ListInsert(MSTree, k, e))  __(4)__ ;  // 插入生成树
    mix_mfset(F, r1, r2);                 // 将两棵树归并为一棵树
}
__(5)__ ;                                 // 继续考查下一条权值最小边
}
DestroySet(F);
}
```

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(共 15 分)

阅读下列说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

在某些系统中，存在非常复杂的对象，可以采用循序渐进的方式将小对象组合成复杂的对象。

以下实例展示了 Builder(生成器)模式。该实例用来建立“文件”，文件内容包括一个标题、一串字符以及一些有项目符号的项目。Builder 类规定组成文件的方法，Director 类利用这个方法产生一份具体的文件。图 7-14 显示了各个类间的关系。

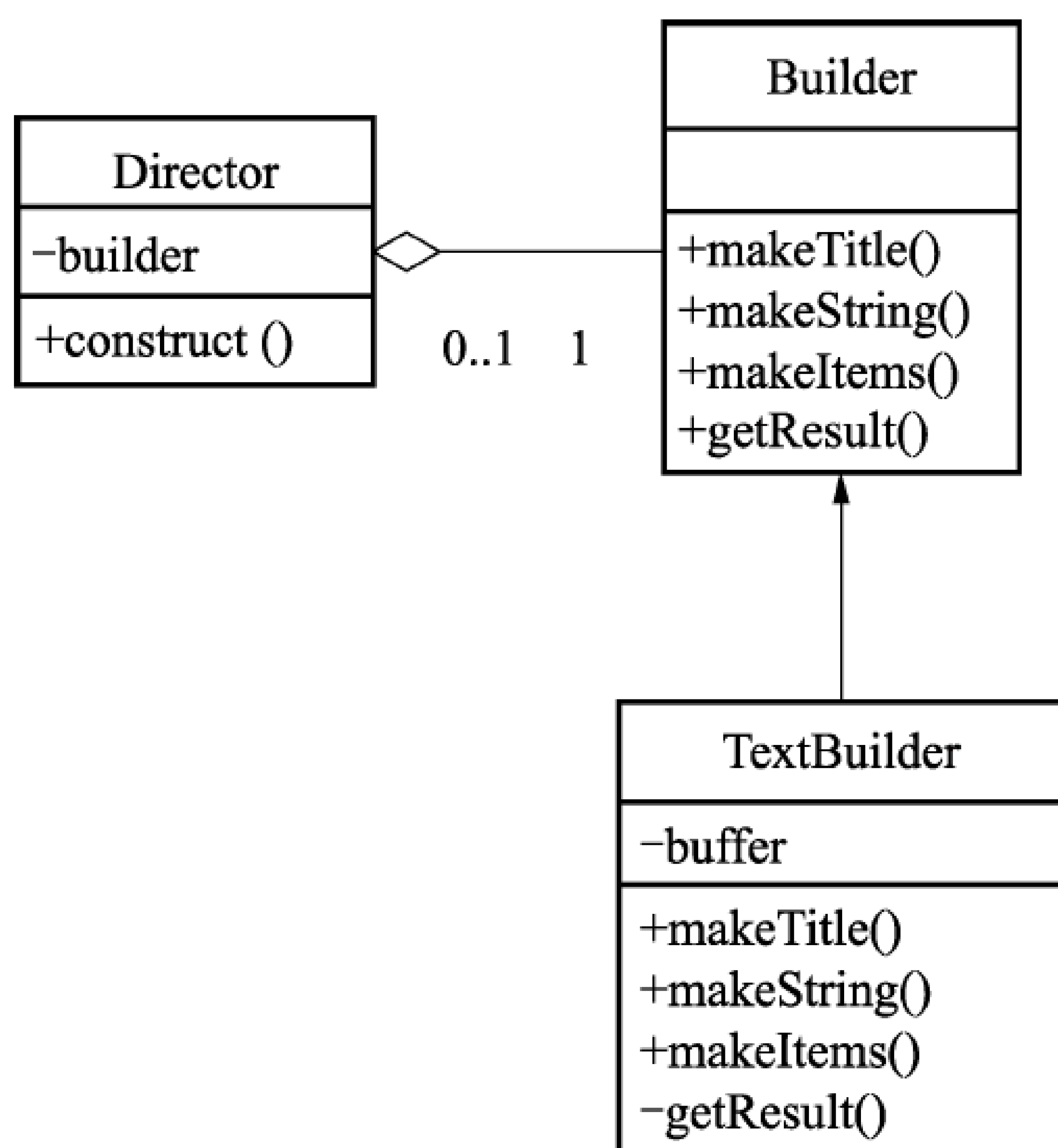


图 7-14　类关系图

以下是 C++语言实现，能够正确编译通过。

【C++程序】

```
class Builder{
public:
    virtual void makeTitle(string title) = 0;
    virtual void makeString(string str) = 0;
    virtual void makeItems( (1)  items) = 0;
    virtual string getResult() = 0;
};

class Director{
private:
     (2)  builder;
public:
    Director(Builder *builder){
        this->builder = builder;
    }
    string construct(){
        vector<string> items;
        items.push_back("早安"); items.push_back("午安");
        builder->makeTitle("Greeting");
        builder->makeString("从早上到白天结束");
        builder->makeItems(items);
        builder->makeString("到了晚上");
         (3) ;//清空 items 向量
        items.push_back("晚安"); items.push_back("好梦");
        builder->makeItems(items);
        return builder->getResult();
    }
};

class TextBuilder:public  (4)  {
private:
    string buffer;
public:
    TextBuilder(){
        buffer = "";
    }
    void makeTitle(string title){
        buffer += "========================================\n";
        buffer += "『" + title + "』\n";
        buffer += "\n";
    }
    void makeString(string str){
        buffer += "■" + str + "\n";
        buffer += "\n";
    }
    void makeItems(vector<string> items){
        vector<string>::iterator it;
        for(it = items.begin(); it != items.end(); it++){
```

```
            buffer += "·" + *it + "\n";
        }
        buffer += "\n";
    }
    string getResult(){
        buffer += "========================\n";
        return buffer;
    }
};

int main()
{
    Director *director = new Director(new TextBuilder());
    string result = (string)director->(5);
    cout<<result;
    return 0;
}
```

试题六(共 15 分)

阅读以下说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

在某些系统中，存在非常复杂的对象，可以采用循序渐进的方式将小对象组合成复杂的对象。

以下实例展示了 Builder(生成器)模式。该实例用来建立“文件”，文件内容包括一个标题、一串字符以及一些有项目符号的项目。Builder 类规定组成文件的方法，Director 类利用这个方法产生一份具体的文件。图 7-15 显示了各个类间的关系。

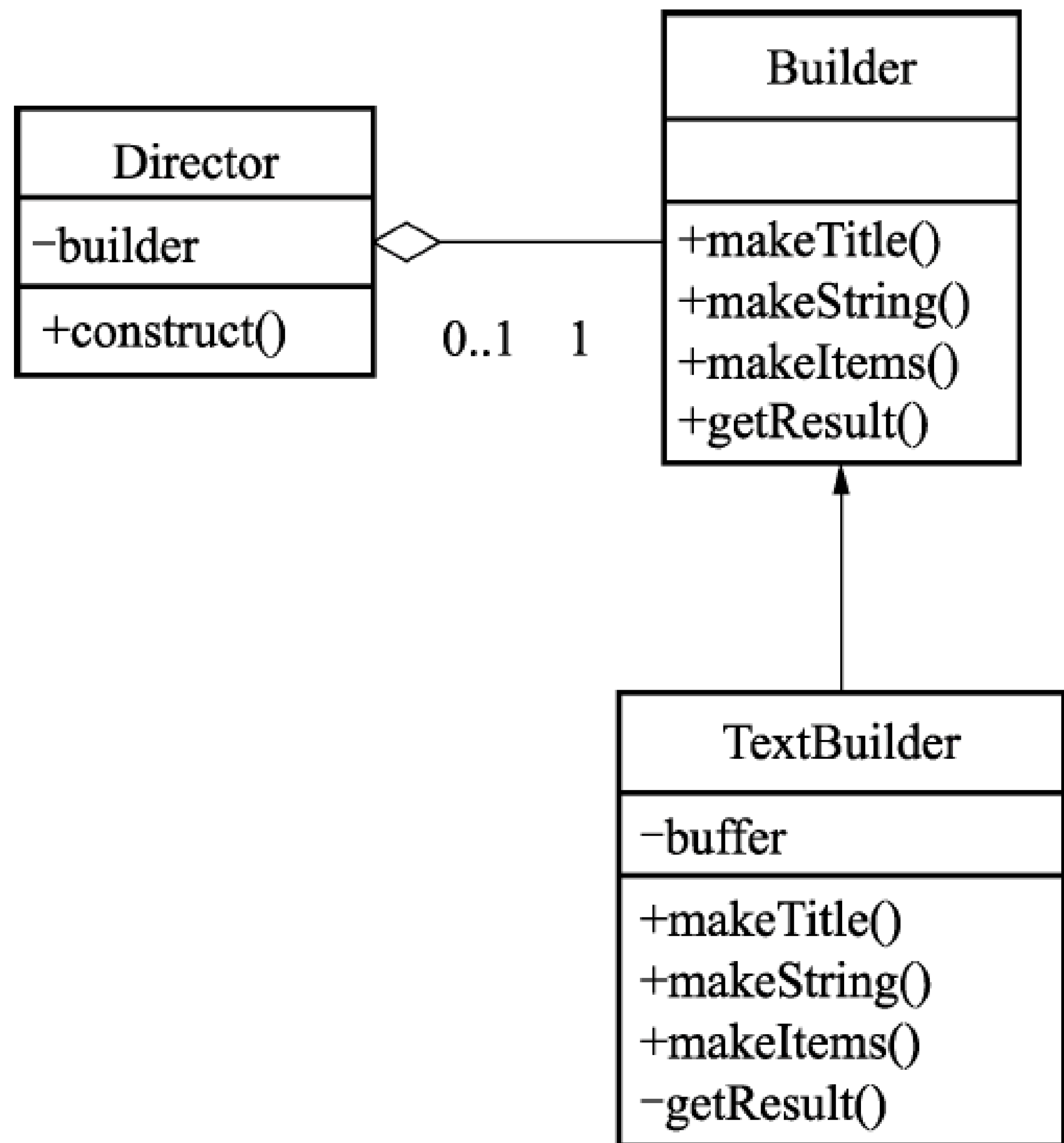
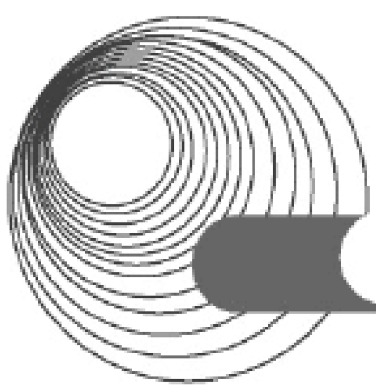


图 7-15　类关系图

以下是 Java 语言实现，能够正确编译通过。



【Java 程序】

```
//Builder.java 文件
public  (1)  class Builder {
    public abstract void makeTitle(String title);
    public abstract void makeString(String str);
    public abstract void makeItems(String[] items);
    public abstract Object getResult();
}

//Director.java 文件
public class Director {
    private  (2)  builder;
    public Director(Builder builder){
        this.builder = builder;
    }
    public Object construct(){
        builder.makeTitle("Greeting");
        builder.makeString("从早上到白天结束");
        builder.makeItems(new String[]{"早安", "午安",});

        builder.makeString("到了晚上");
        builder.makeItems(new String[]{"晚安", "好梦",});
        return builder.getResult();
    }
}

//TextBuilder.java 文件
public class TextBuilder  (3)  Builder {
    private StringBuffer buffer = new StringBuffer();
    public void makeTitle(String title){
        buffer.append("『" + title + "』\n\n ");
    }
    public void makeString(String str){
        buffer.append('■' + str + "\n\n ");
    }
    public void makeItems(String[] items){
        for(int i = 0; i <  (4) ; i++){
            buffer.append('·' + items[i] + "\n");
        }
        buffer.append("\n");
    }
    public Object getResult(){
        return buffer.toString();
    }
}

//Main.java 文件
public class Main {
    public static void main(String[] args) {
        Director director = new Director(new TextBuilder());
```

```
        String result = (String)director. (5) ;
        System.out.println(result);
}
```

试题七(共 15 分)

阅读下列说明、图和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

当一元多项式 $\sum_{i=0}^{N} a_i x^i$ 中有许多系数为零时，可用一个单链表来存储，每个节点存储一个非零项的指数和对应系数。

为了便于运算，用带头节点的单链表存储，头节点中存储多项式中的非零项数，且各节点按指数递减顺序存储。例如，多项式 $8x^5 - 2x^2 + 7$ 的存储结构如图 7-16 所示。

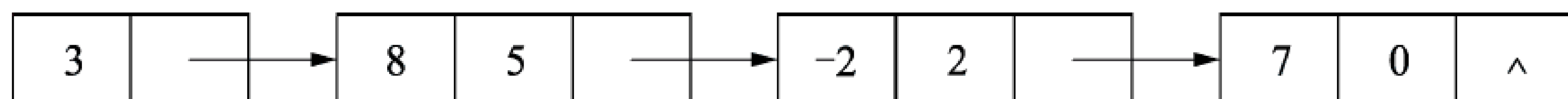


图 7-16　多项式 $8x^5 - 2x^2 + 7$ 的存储结构

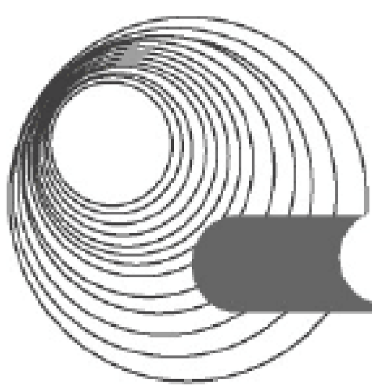
函数中使用的预定义符号如下。

```
#define EPSI 1e-6
struct Node{                /*多项式中的一项*/
    double c;               /*系数*/
    int e;                  /*指数*/
    struct Node *next;
};
typedef struct{             /*多项式头节点*/
    int n;                  /*多项式不为零的项数*/
    struct Node *head;
}POLY;
```

【C++程序】

```
void Del(POLY *C, struct Node *p)
/*若 p 是空指针则删除头节点，否则删除 p 节点的后继*/
{
    struct Node *t;
    /*C 是空指针或 C 没有节点*/
    if(C == NULL || C->head == NULL)return;
    if( (1) ){ /*删除头节点*/
        t = C->head;
        C->head = t->next;
        return;
    }/*if*/
    t = p->next;
    p->next = t->next;
};/*Del*/

void Insert(POLY *C, struct Node *pC)
/*将 pC 节点按指数降序插入到多项式 C 中*/
/*若 C 中存在 pC 对应的指数项，则将系数相加；若其结果为零，则删除该节点*/
```




```
{
    struct Node *t, *tp;
    /*pC 为空指针或其系数近似为零*/
    if(pC == NULL || fabs(pC->c) < EPSI)return;
        if(C->head == NULL){ /*若 C 为空，作为头节点插入*/
        C->head = pC;
        pC->next = NULL;
        C->n++;
        return;
    }/*if*/
    /*若 pC 的指数比头节点的还大，插入到头节点之前*/
    if(pC->e > C->head->e){
        (2) ;
        C->head = pC;
        C->n++;
        return;
    }/*if*/
    (3) ;
    t = C->head;
    while(t!= NULL){
        if(t->e > pC->e){
            tp = t;
            t = t->next;
        }
        else if(t->e == pC->e){ /*C 中已经存在该幂次项*/
            t->c += pC->c; /*系数相加*/
            if(fabs(t->c) < EPSI){ /*系数之和为零*/
                (4) ; /*删除对应节点*/
                C->n--;
            }
            (5) ;
        }
        else t = NULL; /*C 中已经不存在该幂次项*/
    }/*while*/
    if(t == NULL){ /*适当位置插入*/
        pC->next = tp->next;
        tp->next = pC;
        C->n++;
    }/*if*/
};/*Insert*/
```

7.1.3 样卷三

全国计算机技术与软件专业技术资格(水平)考试样卷三

试题一~试题四是必答题

试题一(15 分)

阅读以下说明和 E-R 图，回答问题，将解答写在试卷的对应栏内。

【说明】

设关于图书借阅系统的 E-R 图如图 7-17 所示。图中矩形表示实体，圆表示属性，双圆表示关键字属性，菱形表示实体间的联系。假定已通过下列 SQL 语句建立了基本表。

```
CREATE TABLE Readers
    (Rno CHAR(6) PRIMARY KEY,
    Rname CHAR(20) NOT NULL,
    address CHAR(200),
    phone CHAR(15));
CREATE TABLE Books
    (Bno CHAR(6) PRIMARY KEY,
    name CHAR(50) NOT NULL);
CREATE TABLE Administrators
    (Ano CHAR(6) PRIMARY KEY,
    Aname CHAR(20) NOT NULL);
CREATE TABLE Borrow
    (Rno CHAR(6) NOT NULL,
    Bno CHAR(15) NOT NULL,
    Ano CHAR(6) NOT NULL,
    Bdate DATE,
    Rdate DATE,
    PRIMARY KEY(Rno,Bno,Ano),
    FOREGIN KEY(Rno) REFERENCE Readers(Rno),
    FOREGIN KEY(Bno) REFERENCE Books(Bno),
    FOREGIN KEY(Ano) REFERENCE Administrators(Ano));
```

为了答题的方便，图 7-17 中的实体和属性同时给出了中、英文两种名字，回答问题时只需写出英文名即可。

【系统 E-R 图】

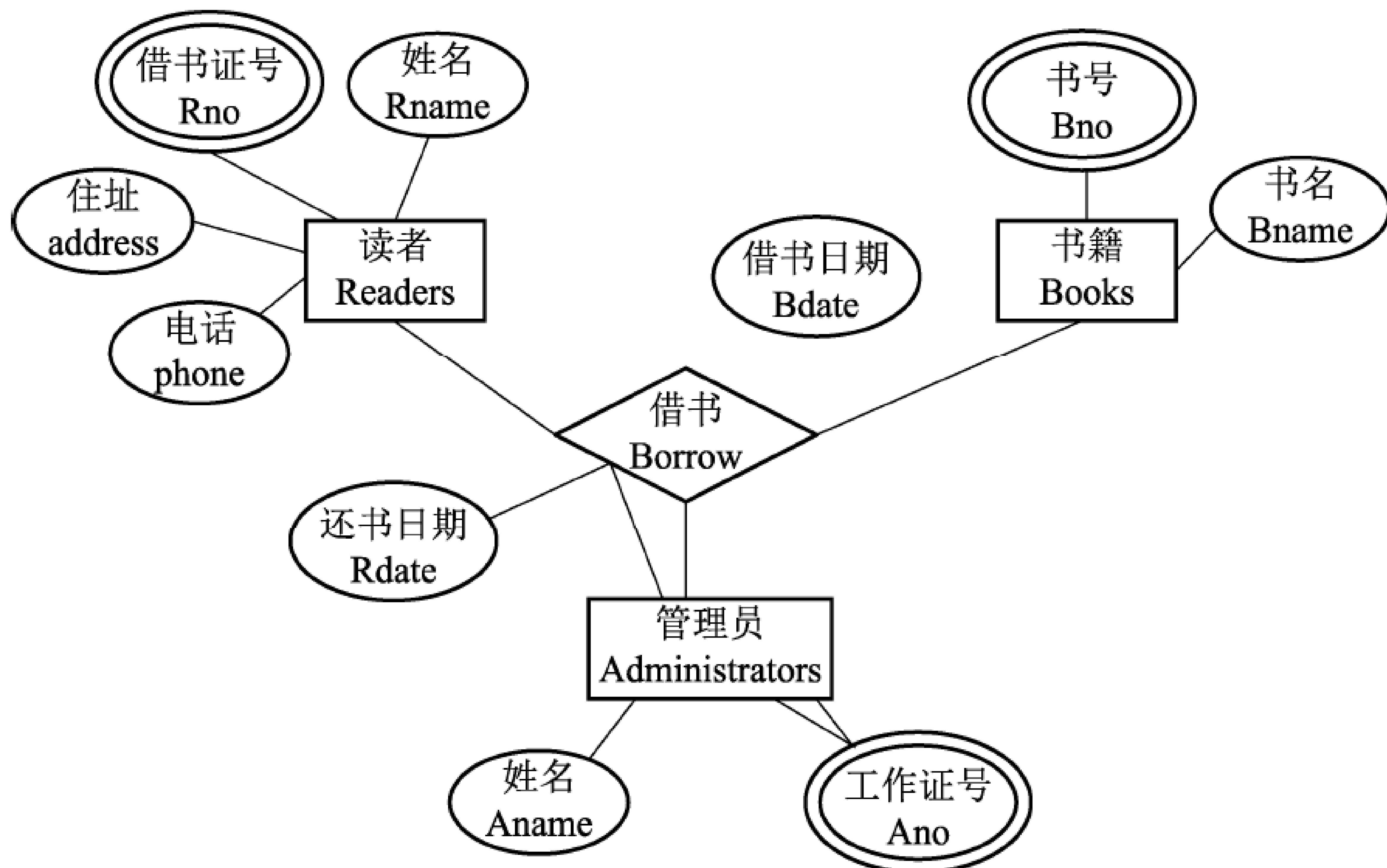
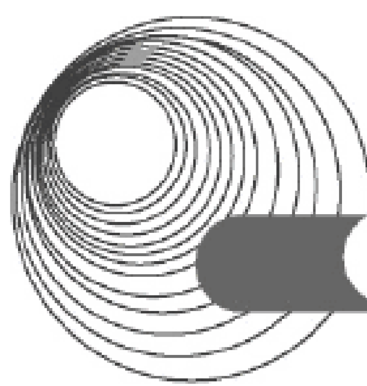


图 7-17　图书借阅管理系统 E-R 图



【问题 1】(13 分)

填充下列 SQL 程序 1~3 中的(1)~(6)空，使它们分别完成相应的功能。

程序 1：查没有借阅过编号为 111111 图书的所有读者名单。

```
SELECT Rno,Rname,address,phone
FROM Readers
WHERE Rno ___(1)___
        (SELECT ___(2)___
         FROM Borrow
         WHERE Bno='111111');
```

程序 2：统计在 2005 年 1 月 1 日借书的读者人数。

```
SELECT ___(3)___
FROM Borrow
WHERE ___(4)___;
```

程序 3：查借书证号为 123456 的读者所借过的所有图书。

```
SELECT Bno,Bname
FROM Books
WHERE ___(5)___
     (SELECT *
      FROM Borrow
       WHERE ___(6)___ AND Rno='123456');
```

【问题 2】(2 分)

对于说明中建立的基本表，是否允许同一读者从同一管理员处多次(两次和两次以上)借阅同一本书？为什么？

试题二(15 分)

阅读下列说明和数据流图，回答问题 1~问题 3。

【说明】

某医院收费系统的主要功能是收取病人门诊的各项费用。系统的收费功能分为 3 个方面：病历收费、挂号收费和根据处方单内容收取检查或药物费用。

(1) 病人初次来该医院看病首先需记录病人基本情况，并购买病历。

(2) 病人看病前要挂号。根据病人的病历和门诊部门(内科、外科等)，系统提供相应的挂号单和处方单，并收取费用。

(3) 病人根据处方单做进一步检查或取药前需交纳各项费用。系统首先根据病人基本情况检查处方单中病历号是否正确，记录合格的处方单并提供收据。

(4) 所有收费都必须依据定价表中的定价来计算，且所有收费都必须写入收费记录中。

医院收费系统的顶层图如图 7-18 所示；医院收费系统的 0 层图如图 7-19 所示。其中，加工 1 子图如图 7-20 所示，加工 3 子图如图 7-21 所示。

假定顶层图是正确的，“定价表”文件已由其他系统生成。

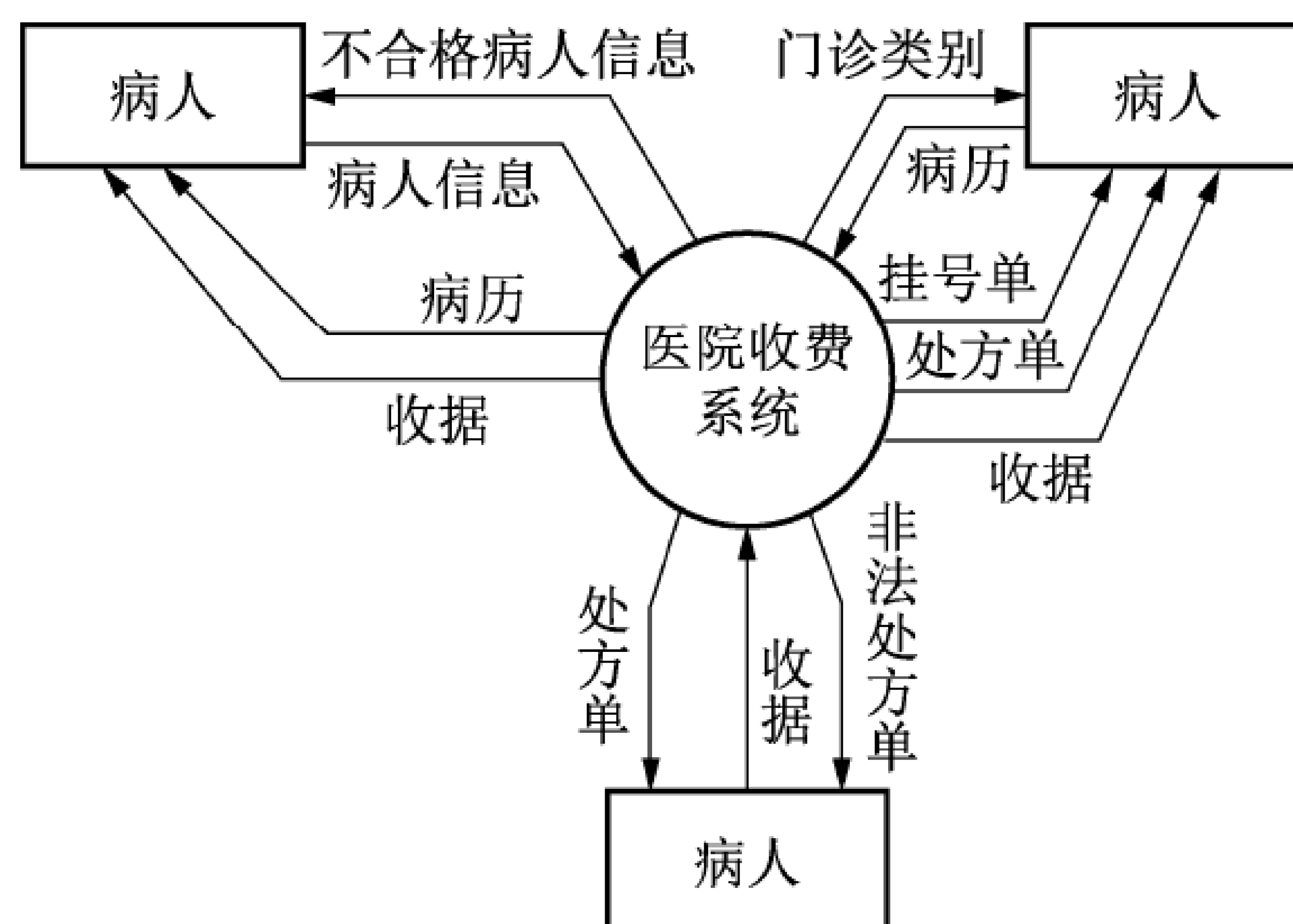


图 7-18　医院收费系统的顶层图

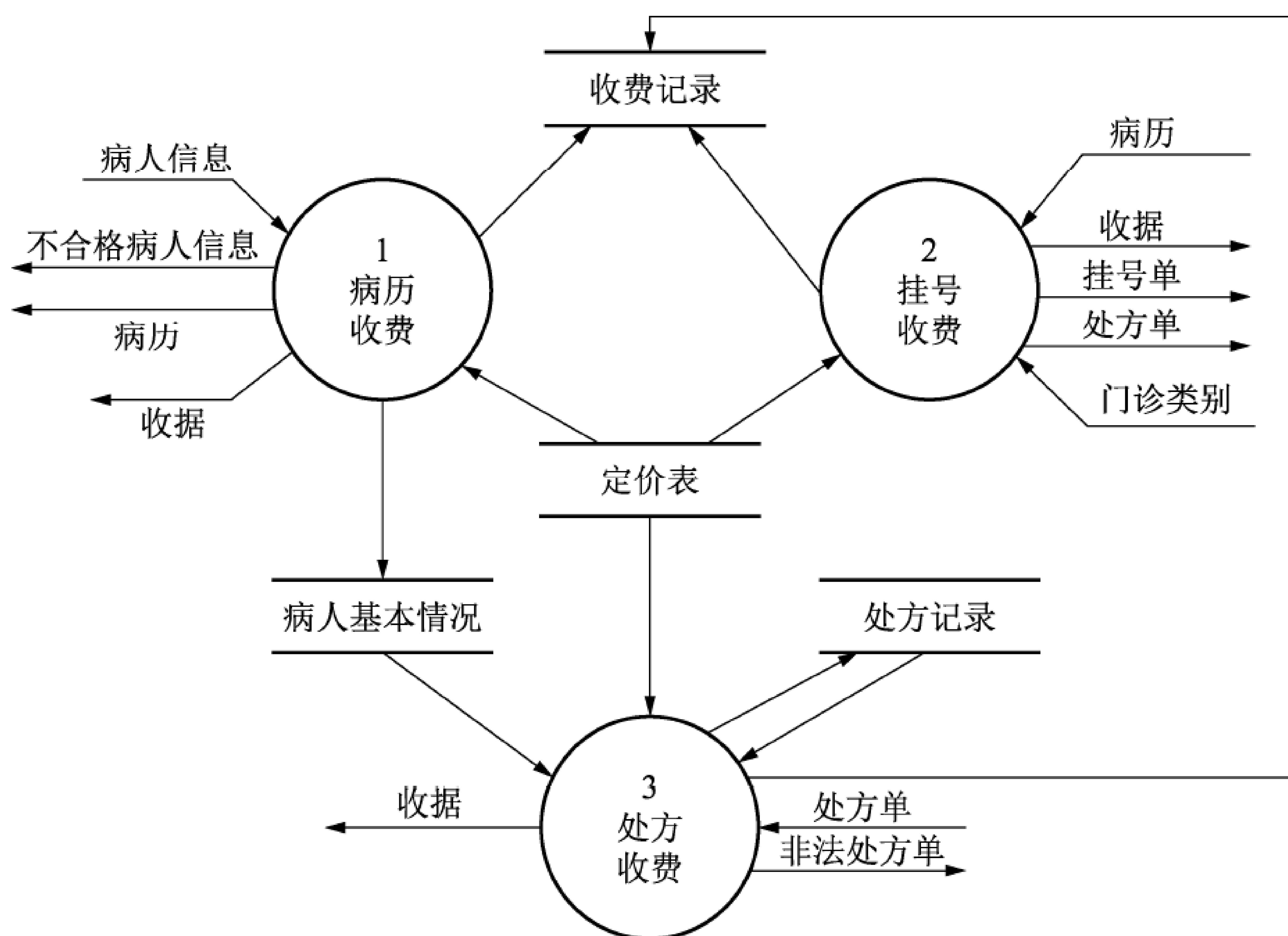


图 7-19　医院收费系统的 0 层图

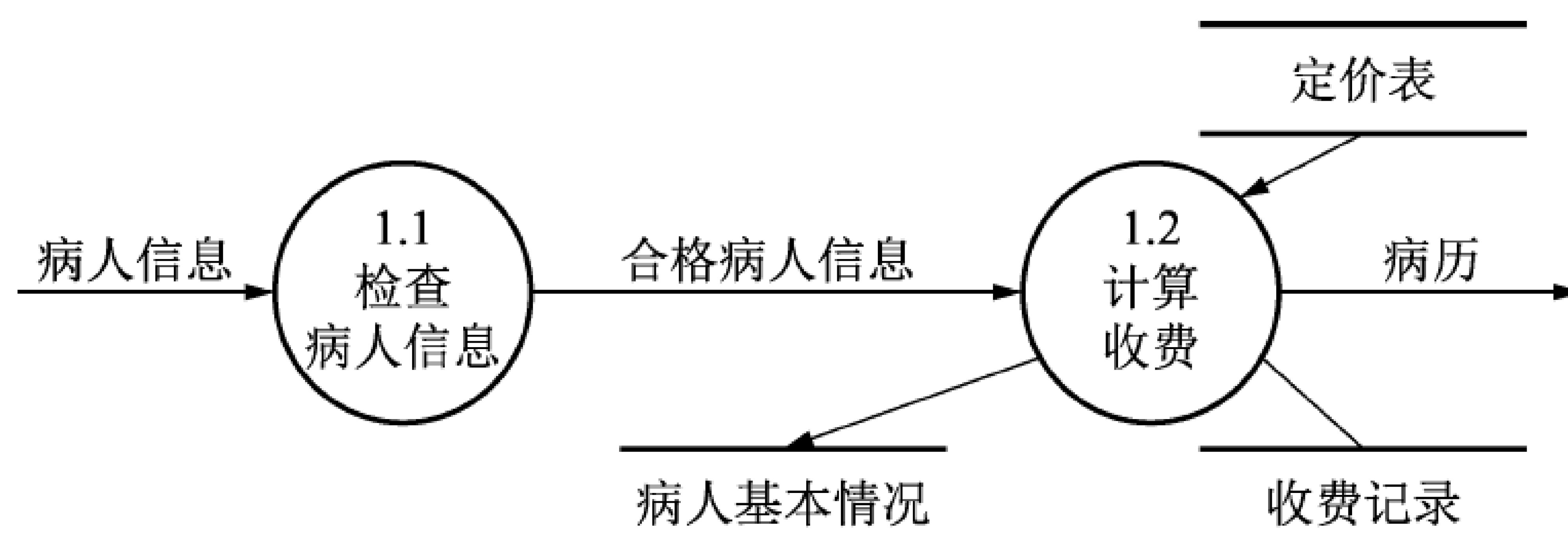
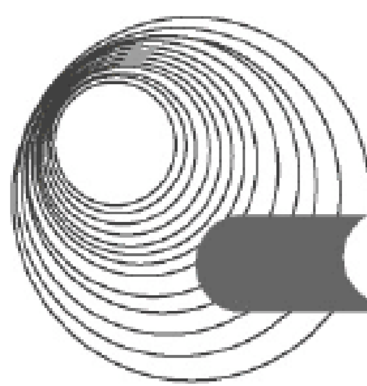


图 7-20　医院收费系统的加工 1 子图



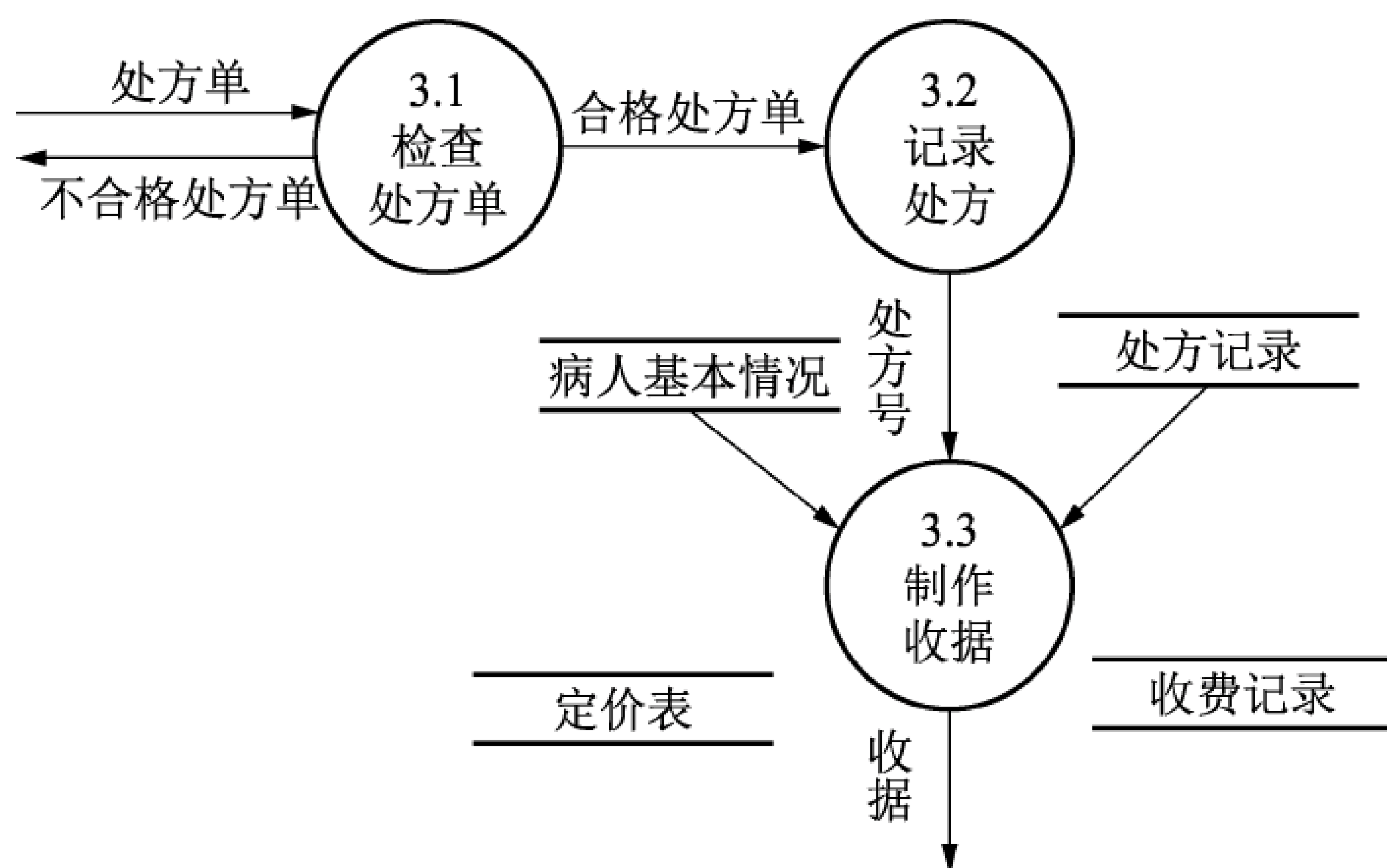


图 7-21　医院收费系统的加工 3 子图

【问题 1】(3 分)

指出哪张图的哪些文件可以不必画出。

【问题 2】(8 分)

图 7-21 中缺少 4 条数据流，请直接在图中添加。

【问题 3】(4 分)

图 7-20 中缺少 2 条数据流，请直接在图中添加。

试题三(15 分)

阅读下列说明及图示，回答问题 1~问题 3。

【说明】

某大学准备开发一个学生课程注册系统，学生可以使用该系统查询新学期将开设的课程和讲课教师情况，选择自己要学习的课程进行登记注册，并可以查询成绩单；教师可以使用该系统查询新学期将开设的课程和选课学生情况，并可以登记成绩单；注册管理员使用该系统进行注册管理，包括维护教师信息、学生信息和课程信息等。

在每个学期的开始，学生可以获得该学期的课程目录表，课程目录表列出每门课程的所有信息，诸如基本信息、教师、开课系和选课条件等。

新学期开始前两周为选课注册时间，在此期间学生可以选课注册，并且允许改变或取消注册申请，开学两周后注册管理员负责关闭课程注册。每个学生可以选择不超过 4 门课程，同时指定 2 门候选课程以备主选课程未选上。每门课程最多不能超过 10 人，最少不能低于 3 人，低于 3 人选课的课程将被取消。一旦学生的注册过程完毕，注册系统将有关信息提交收费系统以便学生付费。如果在实际注册过程中名额已满，系统将通知学生在提交课程表之前予以更改。

在学期结束时，学生可以存取系统查看电子成绩单。由于学生成绩属于敏感信息，系统必须提供必要的安全措施以防非法存取。

【用例图】

学生课程注册系统的用例图如图 7-22 所示。表 7-2~表 7-4 分别为学生课程注册系统的实体类、边界类和控制类。创建课程登记表的协作图如图 7-23 所示。创建课程登记表的时

序图如图 7-24 所示。

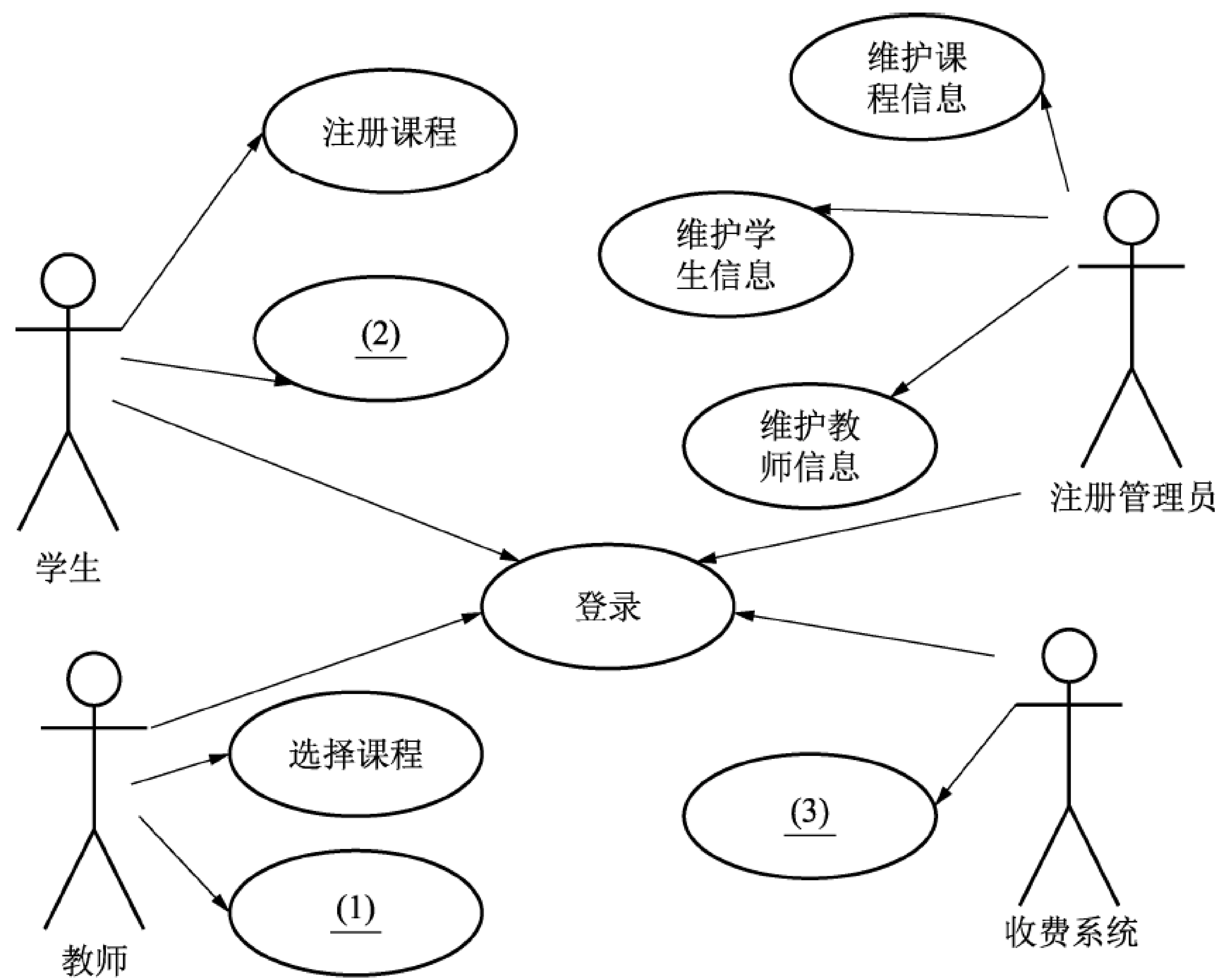


图 7-22　学生课程注册系统的用例图

表 7-2　学生课程注册系统的实体类

实 体 类	说　明
Professor	学校中讲课的教师
Student	学校中注册课程的学生
Schedule	学生在新学期选择登记的课程列表
CourseCatalog	学校所有课程的目录
Course	课程的基本信息
CourseOffering	新学期课程的开设信息，如讲课教师、时间、地点等信息

表 7-3　学生课程注册系统的边界类

边 界 类	说　明
LoginForm	为教师、学生和注册管理员提供登录的操作
RegisterCoursesForm	为学生提供选课注册的操作
ViewReportForm	为学生提供成绩查询的操作
SelectTeachCoursesForm	为教师提供查看学生选课情况的操作
SubmitGradesForm	为教师提供登记成绩的操作
MaintainProfessorsForm	为注册管理员提供维护教师信息的操作

续表

边 界 类	说　明
MaintainStudentsForm	为注册管理员提供维护学生信息的操作
MaintainCoursesForm	为注册管理员提供维护课程信息的操作
CloseRegistrationForm	为注册管理员提供关闭注册的操作
BillingSystemNotice	提供与收费系统的信息交换接口

表 7-4　学生课程注册系统的控制类

控 制 类	说　明
RegisterCoursesControl	负责新学期学生的选课登记
ViewReportControl	负责学生成绩的查询
SelectTeachCoursesControl	负责新学期课程的学生选择情况
SubmitGradesControl	负责学生成绩的登记
CloseRegistrationControl	负责关闭课程注册

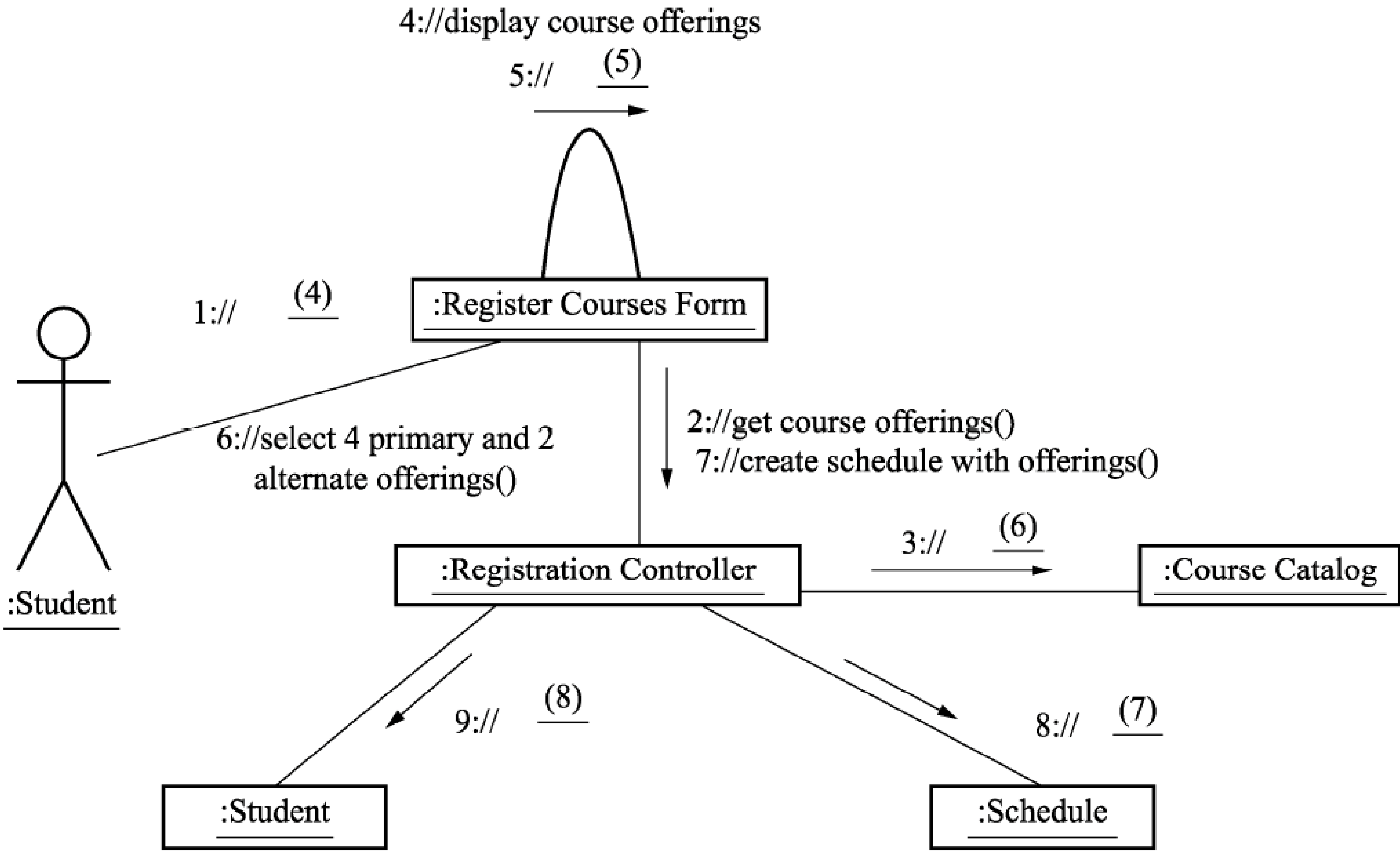


图 7-23　创建课程登记表的协作图

【问题 1】(3 分)

在 UML 中，用例代表一个完整的功能，如与角色通信、进行计算或在系统内工作等。请简要说明用例具有哪些特征，并指出用例图中(1)~(3)处表示的内容。

【问题 2】(5 分)

协作图与时序图是同构的，二者表示的都是同样的系统交互活动，只是各自的侧重点不同而已。根据题目提供的信息，指出协作图中(4)~(8)处表示的内容。

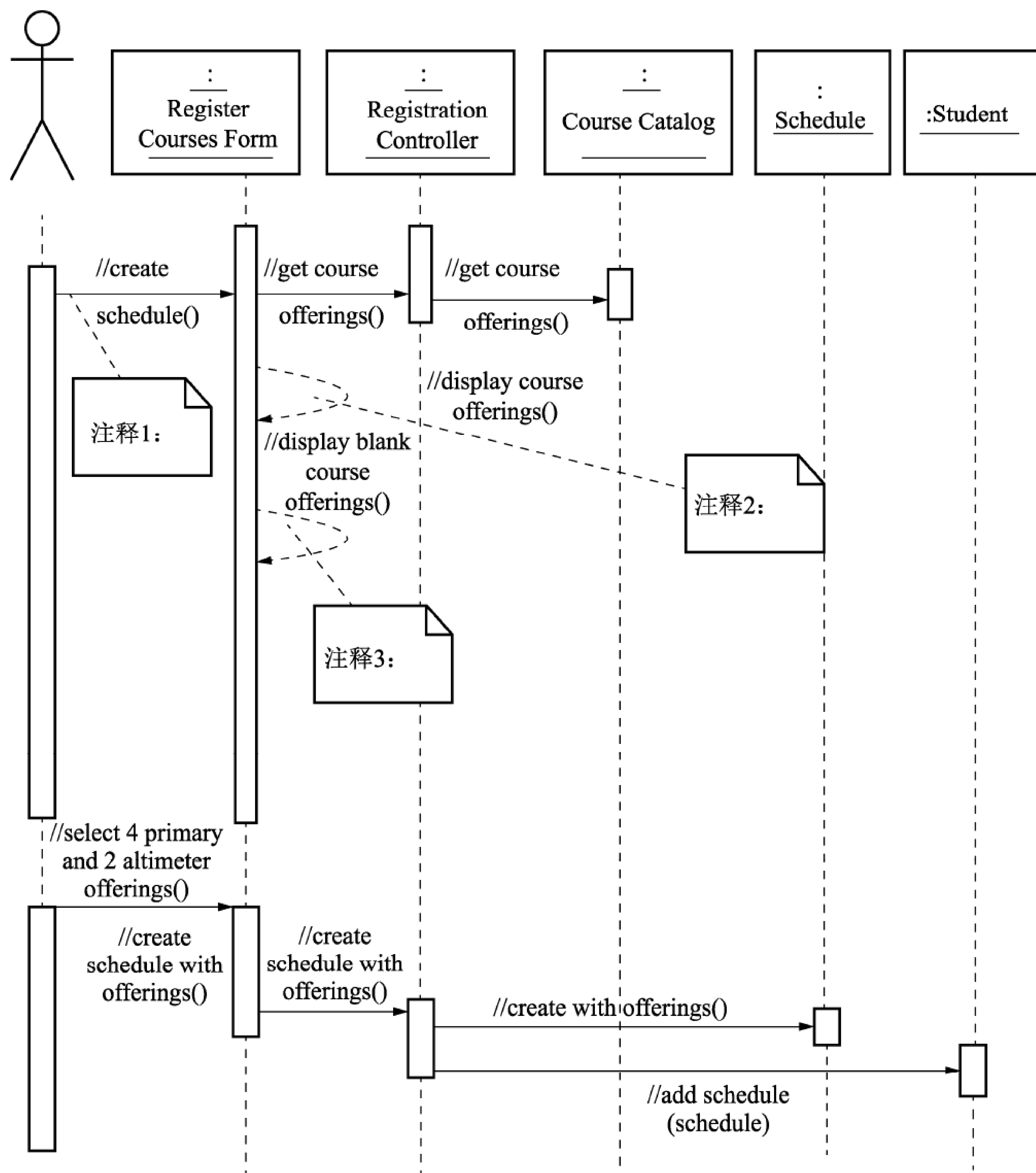


图 7-24　创建课程登记表的时序图

注释 1：学生打算注册新的课程。

注释 2：一张这学期可选择的课程列表。

注释 3：显示一张为学生选课用的空白登记表。

【问题 3】(7 分)

UML 采用 5 个互联的视图来描述软件系统的体系结构，即用例视图(Use-case View)、设计视图(Design View)、进程视图(Process View)、实现视图(Implementation View)和展开视图(Deployment View)。系统模型中每一个视图的内容是由一些图来描述的，UML 中包含用例图、类图、对象图、状态图、时序图、协作图、活动图、组件图、分布图 9 种图。对整个系统而言，其功能由用例图描述，静态结构由类图和对象图描述，动态行为由状态图、时序图、协作图和活动图描述，而物理架构则是由组件图和分布图描述。请分别指出用例

图、类图、对象图、状态图、时序图、协作图、活动图、组件图、分布图的作用。

试题四(15 分)

阅读下列函数说明，将应填入(n)处的子句写在答卷纸的对应栏内。

【函数 1 说明】

函数 compare(SqList A, SqList B)的功能是：设 $A=(a_1,a_2,\cdots,a_m)$和 $B=(b_1,b_2,\cdots,b_n)$均为顺序表，“比较”两个顺序表 A 和 B 的大小。设 A′和 B′分别为 A 和 B 中除去最大共同前缀后的子表(例如，A=(y,x,x,z,x,z)，B=(y,x,x,z,y,x,x,z)，则两者中最大的共同前缀为(y,x,x,z)，在两表中除去最大共同前缀后的子表分别为 A′=(x,z)和 B′=(y,x,x,z))。若 A′=B′=空表，则 A=B；若 A′=空表，而 B′≠ 空表，或者两者均不为空表，且 A′的首元素小于 B′的首元素，则 A<B；否则 A>B。

提示：算法的基本思想为：若 A=B，则 j 增 1，之后继续比较后继元素；否则即可得出比较结果。显然，j 的初值应为 0，循环的条件是 j 不超出其中任何一个表的范围。若在循环内不能得出比较结果，则循环结束时有 3 种可能出现的情况需要区分。

【函数 1】

```
int compare( SqList A, SqList B )
  {
   // 若 A<B，则返回 -1；若 A=B，则返回 0；若 A>B，则返回 1
   j=0;
   while ( j<___(1)___&& j<B.length )
     if ( A.elem[j] < B.elem[j] ) return(-1);
     else if ( A.elem[j] > B.elem[j] ) return(1);
     else___(2)___;
   if ( A.length == B.length ) return (0);
   else if ( A.length < B.length ) return(-1);
   else return(1);
  } // compare
```

函数 1 的时间复杂度是___(3)___。

【函数 2 说明】

函数 exchange_L(SLink &L，int m)的功能是：用尽可能少的辅助空间将单链表中的前 m 个节点和后 n 个节点互换，即将单链表$(a_1,a_2,\cdots,a_m,b_1,b_2,\cdots,b_n)$改变成$(b_1,b_2,\cdots,b_n,a_1,a_2,\cdots,a_m)$。

【函数 2】

```
void exchange_L( SLink &L, int m )
  {
   if (___(4)___&& L->next )          // 链表不空且 m!=0
   {
     p = L->next; k = 1;
     while( k< m && p )               // 查找 a_m 所在节点
     {
       p =___(5)___; ++k;
     }
```

```
        if ( __(6)__ && p->next)          // n!=0 时才需要修改指针
        {
          ha = L->next;                   // 以指针 ha 记 a1 节点的位置
          L->next = p->next;              // 将 b1 节点链接在头节点之后
          p->next = NULL;                 // 设 am 的后继为空
          q = __(7)__;                    // 令 q 指向 b1 节点
          while (q->next) q = __(8)__;    // 查找 bn 节点
          q->next = __(9)__;              // 将 a1 节点链接到 bn 节点之后
        }
      }
    }
```

函数 2 的时间复杂度是__(10)__。

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(15 分，每空 1.5 分)

阅读下列说明和算法，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。

【说明】

算法 1 用来检查文本文件中的圆括号是否匹配。若文件中存在圆括号没有对应的左括号或者右括号，则给出相应的提示信息，如表 7-5 所示。

表 7-5　对应的提示信息

文　件	提示信息
(1+2)	
abc)	缺少对应左括号：第 2 行，第 4 列
((def)gx))	缺少对应左括号：第 3 行，第 10 列
(((h)	
ij)(k	
(1ml)	缺少对应右括号：第 5 行，第 4 列；　第 4 行，第 1 列

在算法 1 中，stack 为一整数栈。算法 1 中各函数的说明如表 7-6 所示。

表 7-6　算法 1 中各函数的说明

函 数 名	函数功能
push(int i)	将整数 i 压入栈 stack 中
pop()	stack 的栈顶元素出栈
empty()	判断 stack 栈是否为空。若为空，函数返回 1；否则函数返回 0
nextch()	读取文本文件中的下一个字符，并返回该字符的 ASCII 值，将字符所在的行号以及字符在行中的位置分别存储到变量 row 和 col 中，若遇到文件结束符，则将变量 EOF 置为 true
kind(charch)	判断字符 ch 是左括号还是右括号，若是左括号则函数返回 1；若是右括号则函数返回 2；若两者都不是则函数返回 0

【算法 1】

```
将栈 stack 置空，置 EOF 为 false
ch←nextch();
while(not EOF)
    k←kind(ch);
    if(k == _(1)_)
        push(_(2)_); push(_(3)_);
    else if(k == _(4)_)
        if(not empty())
            pop(); pop();
        else
            显示错误信息(缺少对应左括号或右括号);
            显示行号 row; 显示列号 col;
        endif
    endif
    ch←nextch();
endwhile
if(not empty())
    显示错误信息(缺少对应左括号或右括号);
    while(not empty())
        row←pop(); col←pop();
        显示行号 row; 显示列号 col;
    endwhile
    endif
```

为了识别更多种类的括号，对算法 1 加以改进后得到算法 2。算法 2 能够识别圆括号、方括号和花括号(不同类型的括号不能互相匹配)。改进后，函数 kind(charch)的参数及其对应的返回值如表 7-7 所示。

表 7-7　函数 kind(charch)的参数及其对应的返回值

ch	(	)	{	}	[	]
返回值	1	2	3	4	5	6

【算法 2】

```
将栈 stack 置空，置 EOF 为 false
ch←nextch();
while(not EOF)
    k←kind(ch);
    if(k > 0)
        if(判断条件 1)
            push(_(5)_); push(_(6)_); push(_(7)_);
        else if(判断条件 2 and 判断条件 3)
            pop(); pop(); pop();
        else
            显示错误信息(缺少对应左括号或右括号);
            显示行号 row; 显示列号 col;
        endif
```

```
        endif
        ch←nextch();
    endwhile
    if(not empty())
        显示错误信息(缺少对应左括号或右括号);
        while(not empty())
            pop(); row←pop(); col←pop();
            显示行号 row; 显示列号 col;
        endwhile
    endif
```

【问题 1】

请将算法 1 和算法 2 中(1)~(7)处补充完整。

【问题 2】

请从下面的选项中选择相应的判断逻辑填补算法 2 中的“判断条件 1”~“判断条件 3”。注意，若“判断条件 2”的逻辑判断结果为假，就无须对“判断条件 3”进行判断。

(a) 字符是括号；(b) 字符是左括号；(c) 字符是右括号；(d) 栈空；(e) 栈不空；

(f) 栈顶元素表示的是与当前字符匹配的左括号；

(g) 栈顶元素表示的是与当前字符匹配的右括号。

试题六(共 15 分)

阅读下列函数说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

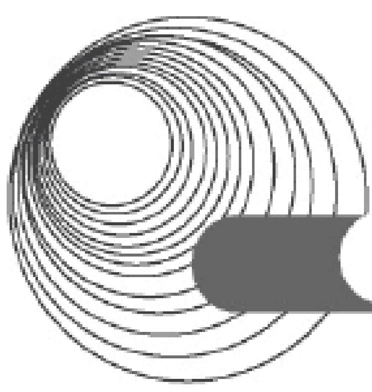
【说明】

任何一种程序都是为了解决问题而撰写的，解决问题时需要实现一些特定的运算法则。在策略(Strategy)模式下，可以更换实现算法的部分而不留痕迹，切换整个算法，简化改为采用其他方法来解决同样问题。

以下是一个“剪刀石头布”游戏。猜拳时的“策略”有两种方法：第一种是“猜赢后继续出同样的招式”(WinningStrategy)；第二种是“从上一次出的招式中，以概率分配方式求出下一个招式的概率”(ProbStrategy)。程序中定义了 Hand 类表示猜拳时的“手势”，类内部以 0(石头)、1(剪刀)、2(布)来表示。Hand 类的实例只会产生 3 个。以下是 C++语言实现，能够正确编译通过。

【C++程序】

```
class Hand{
private:
    int handvalue;
    static Hand *hand0;
    static Hand *hand1;
    static Hand *hand2;
__(1)__:
    Hand(int handvalue){
        this->handvalue = handvalue;
    }
public:
    __(2)__ Hand* getHand(int handvalue){
        /*省略具体实现*/
    }
```




```
};
Hand *Hand::hand0 = new Hand(0);
Hand *Hand::hand1 = new Hand(1);
Hand *Hand::hand2 = new Hand(2);

class Strategy{
public:
    ___(3)___Hand* nextHand() = 0;
};
class WinningStrategy:public Strategy{
private:
    bool won;
    Hand *prevHand;
public:
    WinningStrategy(){
        won = false;
    }
    Hand* nextHand(){
        if(!won){
            prevHand = Hand::getHand(rand()%3);
        }
        return prevHand;
    }
};

class ProbStrategy:public Strategy{
public:
    Hand* nextHand(){
        int handvalue = 0;
        /*省略具体实现*/
        return Hand::getHand(handvalue);
    }
};

class Player{
private:
    string name;
    Strategy* strategy;
public:
    Player(string name,___(4)___strategy){
        this->name = name;
        this->strategy = strategy;
    }
    Hand *nextHand(){//向战略请示手势
        return ___(5)___;
    }
};
```

试题七(共 15 分)

阅读以下说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

任何一种程序都是为了解决问题而撰写的，解决问题时需要实现一些特定的运算法则。在策略(Strategy)模式下，可以更换实现算法的部分而不留痕迹，切换整个算法，简化改为采用其他方法来解决同样问题。

以下是一个“剪刀石头布”游戏。猜拳时的“策略”有两种方法：第一种是“猜赢后继续出同样的招式”(WinningStrategy)；第二种是“从上一次出的招式中，以概率分配方式求出下一个招式的概率”(ProbStrategy)。程序中定义了 Hand 类表示猜拳时的“手势”，类内部以 0(石头)、1(剪刀)、2(布)来表示。Hand 类的实例只会产生 3 个。

以下是 Java 语言实现，省略了不相关属性及方法，方法实现语句亦有所省略，能够正确编译通过。

【Java 程序】

```
//Hand.java 文件
public class Hand {
    public static final int HANDVALUE_GUU = 0; //石头
    public static final int HANDVALUE_CHO = 1; //剪刀
    public static final int HANDVALUE_PAA = 2; //布
    public static final Hand[] hand = {
            new Hand(HANDVALUE_GUU),
            new Hand(HANDVALUE_CHO),
            new Hand(HANDVALUE_PAA),
            };
    private int handvalue;
    ___(1)___Hand(int handvalue){
        this.handvalue = handvalue;
    }
    public___(2)___Hand getHand(int handvalue){//从值取得对象实例
        return hand[handvalue];
    }
}

//Strategy.java 文件
public interface Strategy {
    public___(3)___Hand nextHand();
}

//ProbStrategy.java 文件
import java.util.Random;
public class ProbStrategy implements Strategy {
    public Hand nextHand(){
        int handvalue = 0;
        /*省略具体实现*/
        return Hand.getHand(handvalue);
    }
}
//WinningStrategy.java 文件
import java.util.Random;
public class WinningStrategy implements Strategy {
```

```
        /*省略了不相关属性*/
        public Hand nextHand(){
            if(!won){
                prevHand = Hand.getHand(random.nextInt(3));
            }
            return prevHand;
        }
    }

    //Player.java 文件
    public class Player {
        private String name;
        private Strategy strategy;
        public Player(String name, ___(4)___ strategy){
            this.name = name;
            this.strategy = strategy;
        }
        public Hand nextHand(){//向战略请示手势
            return ___(5)___;
        }
    }
```

7.1.4 样卷四

全国计算机技术与软件专业技术资格(水平)考试样卷四

试题一~试题四是必答题

试题一(15 分)

阅读以下说明和数据流图，回答问题 1~问题 3。

【说明】

某直达列车车票预售系统接受顾客的订票、取票和售票处工作人员的查询业务。

(1) 顾客为了提前订票，可向系统提供个人信息及其期望订购的车次及日期，系统根据个人信息是否齐全以及车次是否正确来判断订票单是否合格。对于合格的订票单系统，如果相应的车次有剩余票，则记录顾客个人信息及订票信息，并向顾客提供取票单。

(2) 到了可以取票的时间，顾客向系统提供取票单，在检查单据合格的情况下，系统向顾客提供火车票。

(3) 售票处的工作人员可以利用系统查询各车次车票的售票情况。

该直达列车车票预售系统的分层数据流图中部分数据流和文件的组成如下。

- 文件：

  火车时刻表=车次+开车时间+到站时间+起始站+终点站+上铺票价+下铺票价；

  订票信息表=车次+车票日期+旅客身份证号+座位号+是否领票；

  旅客信息表=旅客身份证号+姓名+性别+联系电话；

座位表=车次+座位号。

- 数据流：

  订票单=旅客姓名+性别+身份证号+联系电话+车次+车票日期；

  车票=车次+起始站+终点站+开车日期+开车时间+座位号+票价。

  假定顶层图是正确的，“火车时刻表”和“座位表”文件已由其他系统生成。

【问题 1】(2 分)

指出图 7-25~图 7-28 中哪张图的哪个文件可以不必画出。

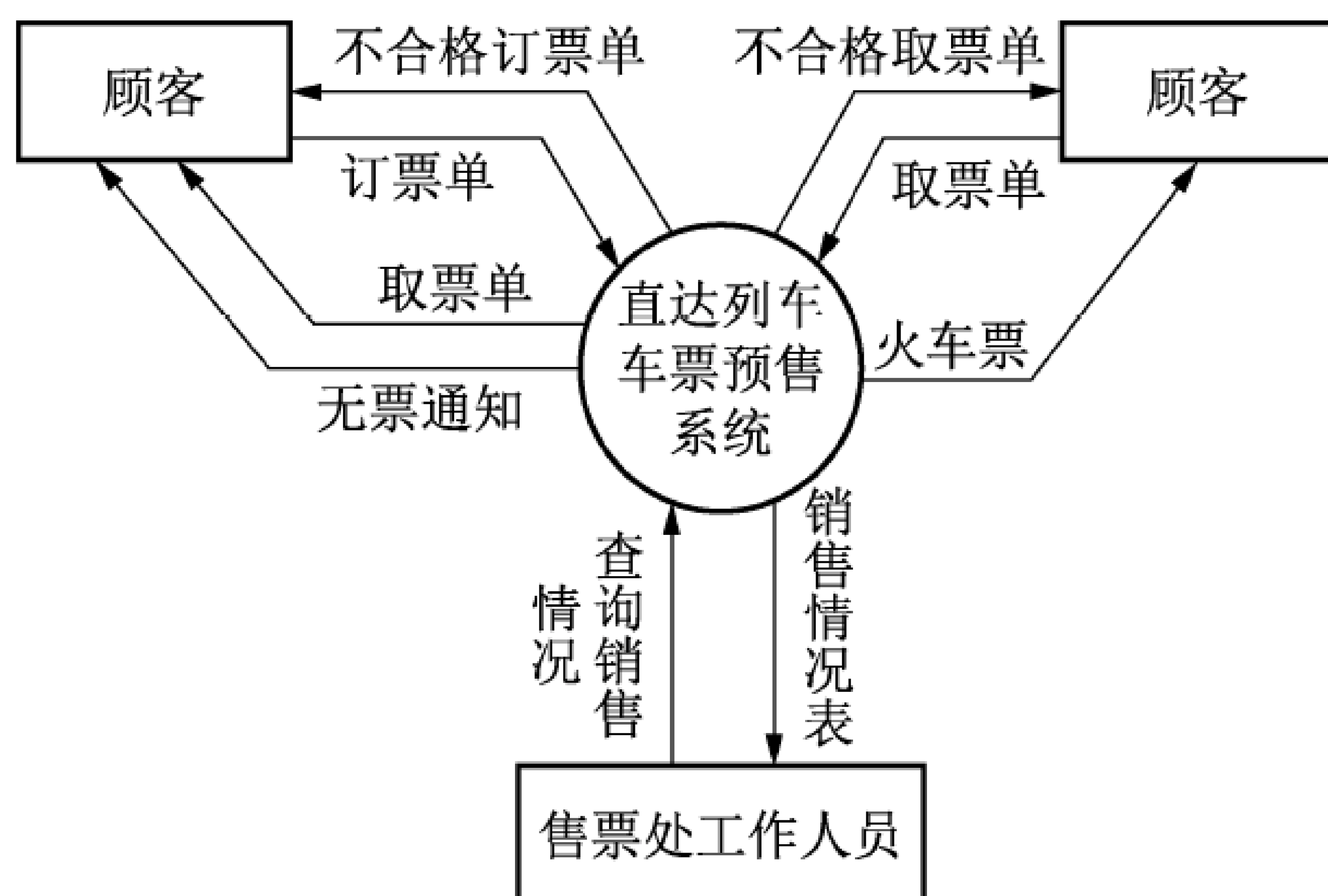


图 7-25　直达列车车票预售系统顶层图

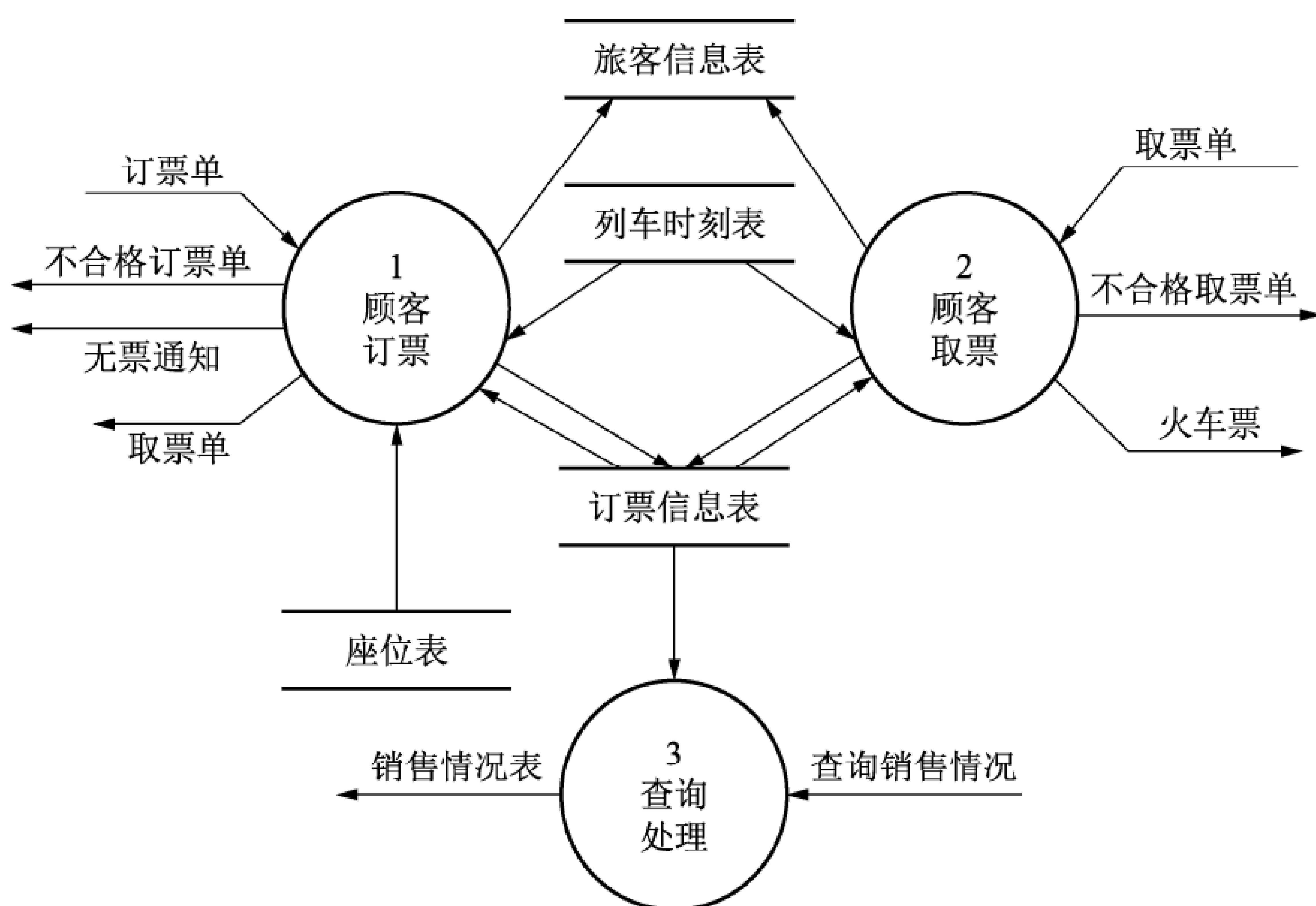


图 7-26　直达列车车票预售系统 0 层图

【问题 2】(8 分)

指出图 7-27 和图 7-28 中错误的数据流。

【问题 3】(5 分)

根据题中说明和数据流图分析，加工 3“查询处理”是否可以查询出剩余票的信息？为什么？

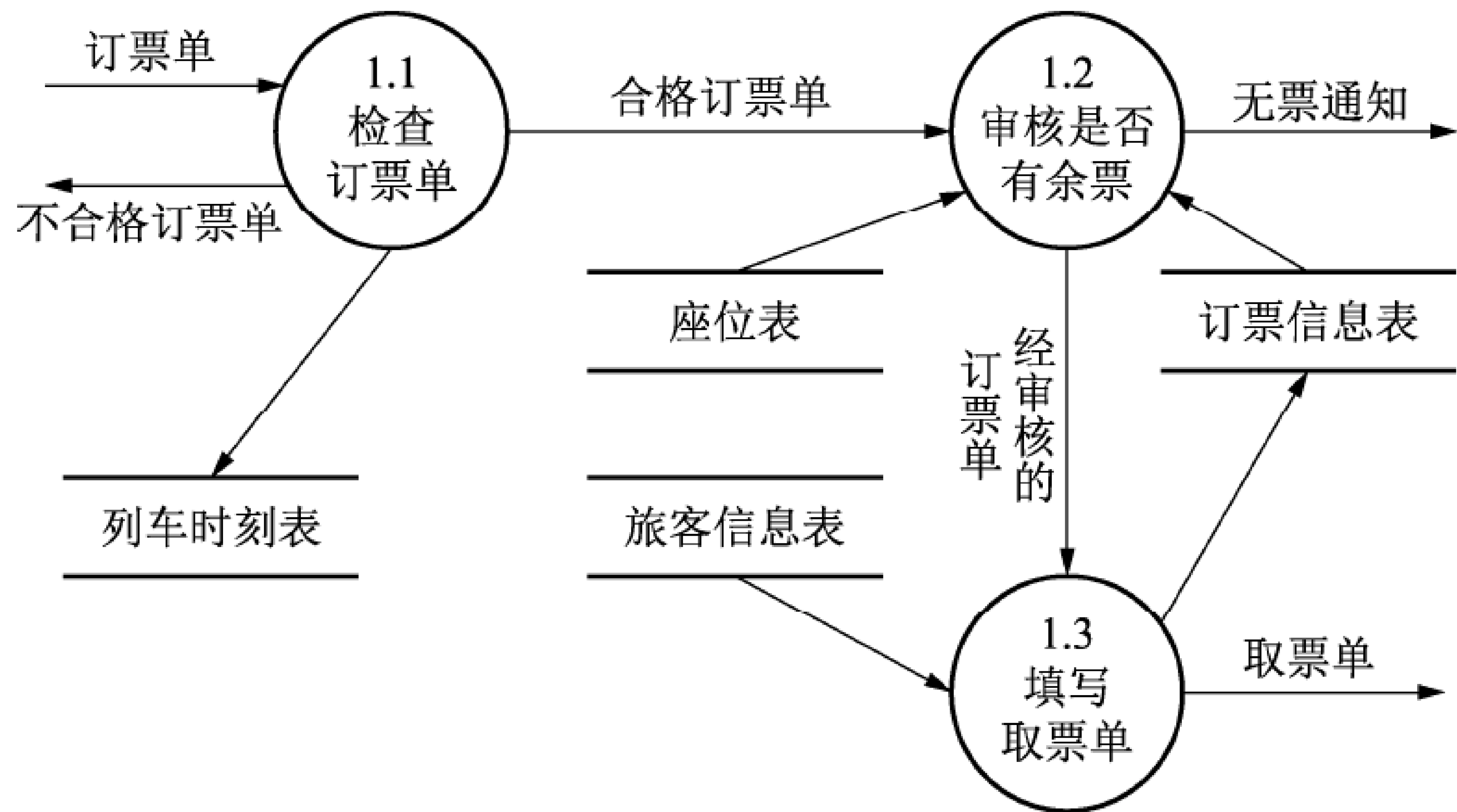


图 7-27　直达列车车票预售系统加工 1 子图

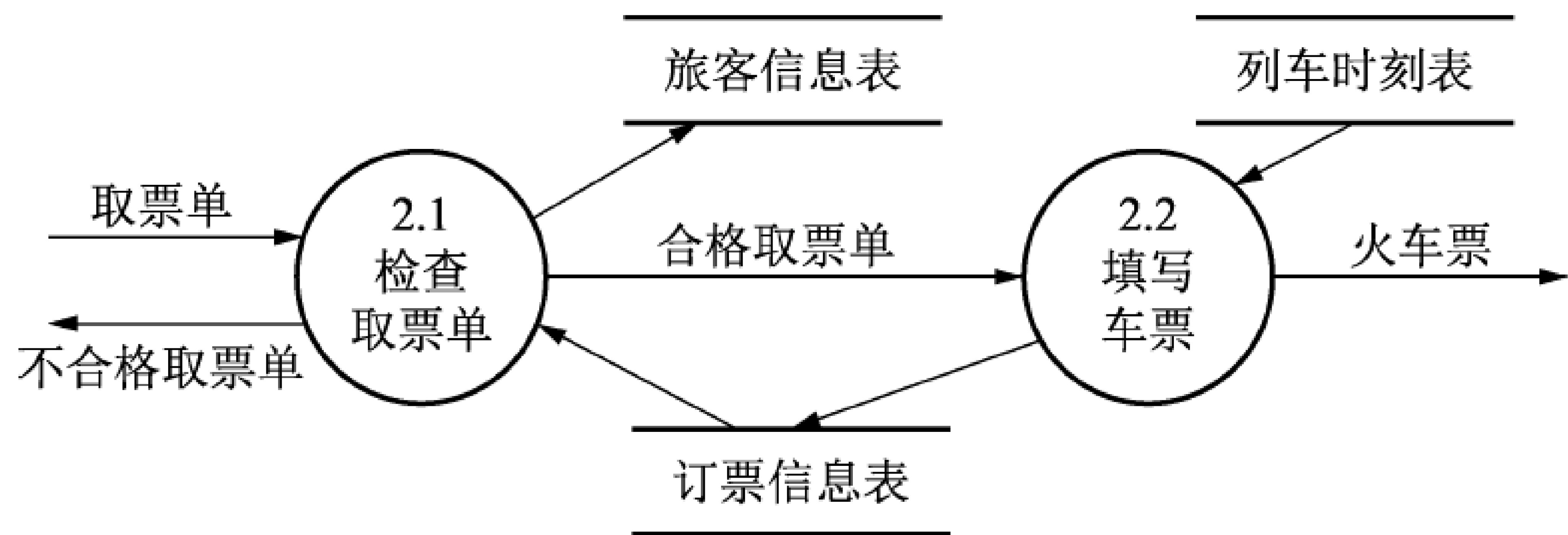


图 7-28　直达列车车票预售系统加工 2 子图

试题二(15 分)

阅读下列说明，回答问题 1~问题 4，将解答填入答题纸的对应栏内。

【说明】

甲公司的经营销售业务目前是手工处理的，随着业务量的增长，准备采用关系数据库对销售信息进行管理。经销业务的手工处理主要涉及 3 种表：订单、客户表和产品表，如图 7-29 所示。

为了用计算机管理销售信息，甲公司提出应达到以下要求：产品的单价发生变化时，应及时修改产品表中的单价数据。客户购货计价采用订货时的单价。订货后，即使单价发生变化，计算用的单价也不变。

订　　单

客户代码：　　　　　　　　订单号：

客户名：　　　　　　　　　订货日期：

订货序号	产品代码	产品名称	数　量	单　价	小　计

总金额：

图 7-29　订单、客户表和产品表

客　户　表

客户代码	客 户 名	地　址	电　话

产　品　表

产品代码	产　品	名　称	单　价

图 7-29　订单、客户表和产品表(续)

在设计数据库时，经销部的王先生建立了如图 7-30 所示的数据模型。

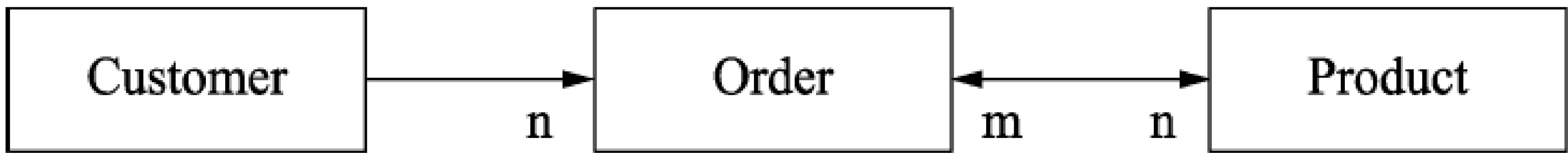


图 7-30　数据模型

其中，方框表示实体，单向箭头表示一对多的联系，双向箭头表示多对多的联系。

由于上述模型对建立关系数据库是不合适的，因此王先生又修改了数据模型，并设计了以下几个关系(带下画线的数据项是关键项，最后一个关系中没有指出关键项)：

Customer(**<u>CustomerNo</u>**, CustomerName, Address, Phone)；

Product(**<u>ProductNo</u>**, ProductName, UnitPrice)；

Order(**<u>OrderNo</u>**, CustomerNo, Date)；

OrderDetail(OrderNo, ProductNo, Quantity)。

【问题 1】

请按说明中的要求画出修改后的数据模型。

【问题 2】

(1) 说明中的几个关系仍无法实现甲公司的要求，为什么？

(2) 需要在哪个关系中增加什么数据项才能实现这个要求？

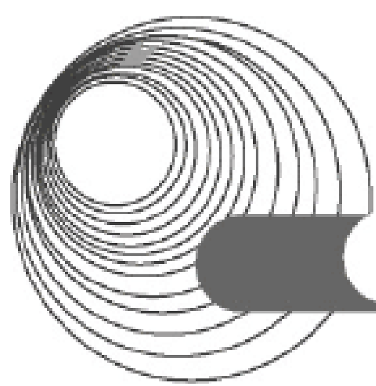
【问题 3】

写出 OrderDetail 中的关键项。

【问题 4】

以下 SQL 语句用于查询没有订购产品代码为“1K10”的产品的所有客户名。请填补其中的空缺。

```
SELECT CustomerName FROM Customer  (1)
WHERE   (2)
(SELECT  *  FROM  OrderDetail  B, Order  C
```




```
WHERE  B. ProductNo = C.ProductNo
AND  B. ProductNo = '1K10'
AND  C. CustomerNo = A.CustomerNo)
```

试题三(15 分)

阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

一个新的音像商店准备向比较广泛的人群出租录像带和光碟。该商店的管理者决定在计算机系统的支持下来运作。

音像商店在货架上存放着题材广泛的当前流行的电影库。由于同一个电影片名可能由于不同的导演而有不同的版本，因此电影用电影代码区分，而不用电影片名；同一个版本有多份副本，因此音像制品用一个唯一的编号标识。某个特定的电影可以存放在录像带或光碟上，录像带和光碟的租金不同。录像带要么是 Beta 格式要么是 VHS 格式；光碟为 DVD 格式，容量比较大，一张光碟可以存储同一电影片名的不同版本。每个电影都有特定的租用期(用天表示)，并带有在租用期内的租金。音像商店必须能够立即回答关于某个电影的库存和有多少供租用的带子或光碟的问题。

音像商店的店员负责订购音像、联系客户、音像上架，并对客户的询问给出答复。

该系统采用面向对象方法开发，系统中的类及类之间的关系用 UML 类图表示。图 7-31 是该系统的用例图；图 7-32 是该系统的类图的一部分。

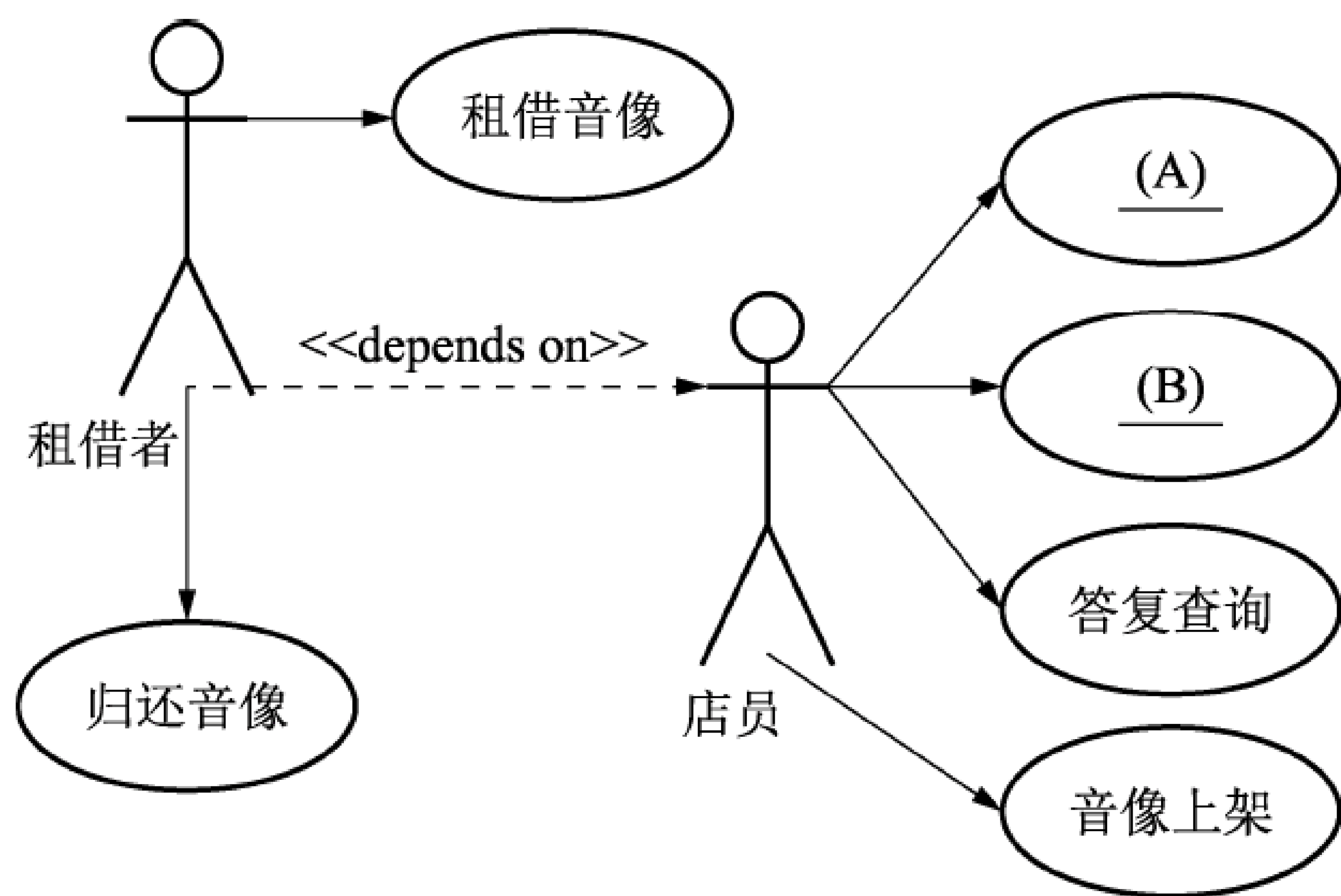


图 7-31　系统用例图

【问题 1】(3 分)

根据题意，给出“电影”类的主要属性。

【问题 2】(4 分)

根据题意，指出图 7-31 中缺失的用例。

【问题 3】(8 分)

根据题意，补充图 7-32 所示的类图中缺失的类之间的关系，用 UML 表示法表示。要求标出重复度。在 UML 中，重复度(Multiplicity)定义了某个类的一个实例可以与另一个类的多少个实例相关联。通常把它写成一个表示取值范围的表达式或者一个具体的值。

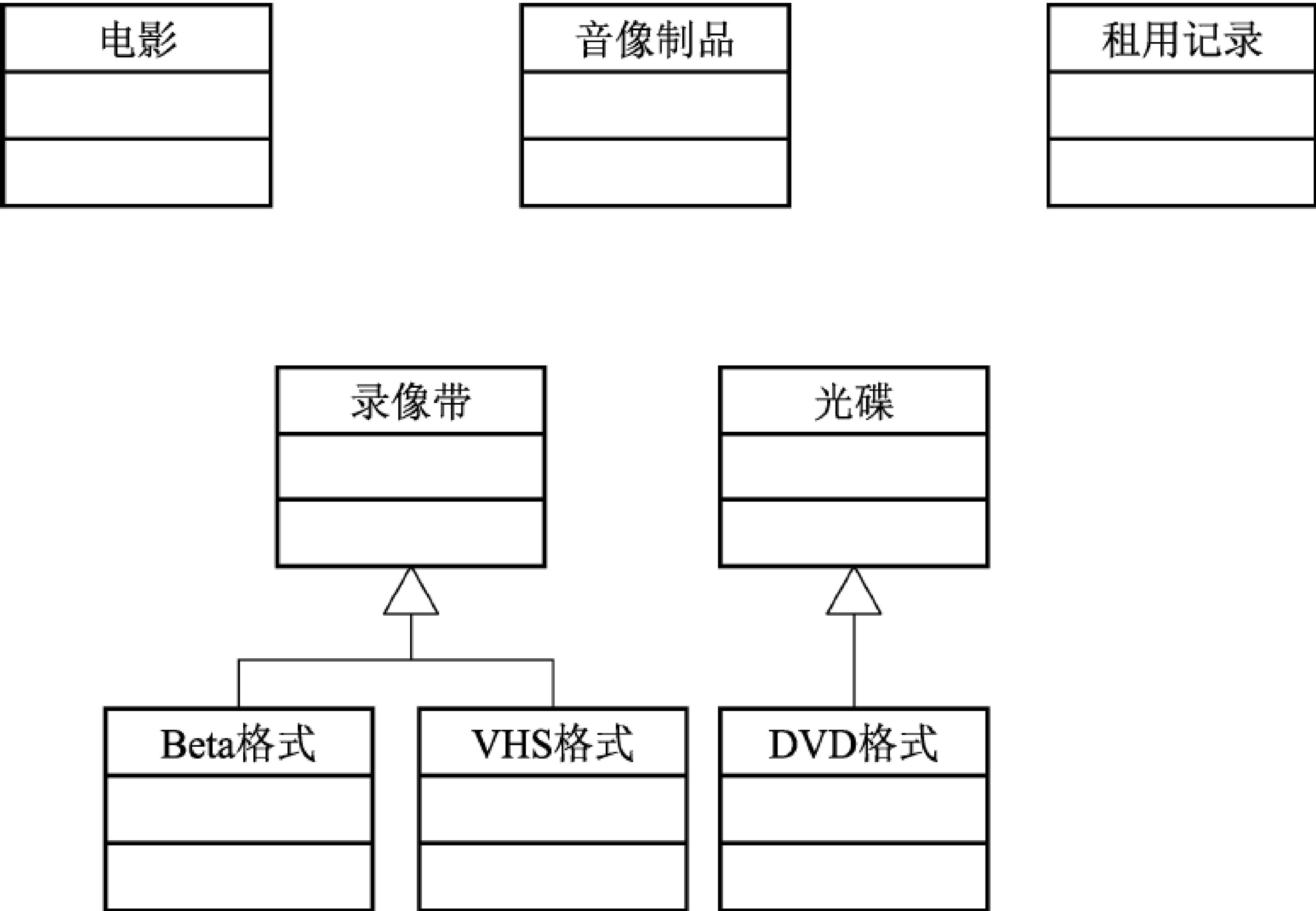


图 7-32　类图的一部分

试题四(15 分)

阅读下列程序说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【程序说明】

“背包问题”的基本描述是：有一个背包，能盛放的物品总重量为 S，设有 n 件物品，其重量分别为 $w_1, w_2, \cdots, w_n$，希望从 n 件物品中选择若干件物品，所选物品的重量之和恰能放入该背包，即所选物品的重量之和等于 S。

以下程序均能求得“背包问题”的一组解，其中程序 1 是“背包问题”的递归解法，而程序 2 是“背包问题”的非递归解法。

【程序 1】

```
#include <stdio.h>
#define N 7
#define S 15
int w[N+1] = {0,1,4,3,4,5,2,7};
int knap(int s, int n)
{
    if(s == 0) return 1;
    if(s < 0 || (s > 0 && n < 1)) return 0;
    if( (1) ){
        printf("%4d", w[n]);
        return 1;
    }
    return  (2) ;
}
main()
{
    if(knap(S,N))printf("OK!\n");
    else printf("NO!\n");
}
```

【程序 2】

```
#include <stdio.h>
#define N 7
#define S 15
typedef struct{
    int s;
    int n;
    int job;
}KNAPTP;
int w[N+1] = {0,1,4,3,4,5,2,7};
int knap(int s, int n);
main()
{
    if(knap(S, N)) printf("OK!\n" );
    else printf("NO!\n");
}
int knap(int s, int n)
{
    KNAPTP stack[100], x;
    int top, k, rep;
    x.s = s; x.n = n;
    x.job = 0;
    top = 1; stack[top] = x;
    k = 0;
    while( (3) ){
        x = stack[top];
        rep = 1;
        while(!k && rep){
            if(x.s == 0) k = 1; /*已求得一组解*/
            else if(x.s < 0 || x.n <= 0) rep = 0;
            else{
                x.s =  (4) ;
                x.job=1;
                 (5) = x;
            }
        }
        if(!k){
            rep = 1;
            while(top >= 1 && rep){
                x = stack[top--];
                if(x.job == 1){
                    x.s += w[x.n+1];
                    x.job = 2;
                    stack[++top] = x;
                     (6) ;
                }
            }
        }
```

```
    }
    if(k){    /*输出一组解*/
        while(top >= 1){
            x = stack[top--];
            if(x.job == 1) printf ("%4d\t", w[x.n+1]);
        }
    }
    return k;
}
```

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(15 分，每空 1.5 分)

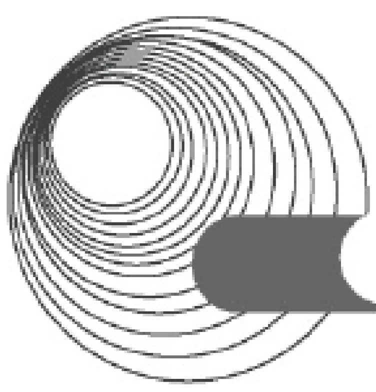
阅读下列程序说明和 C 程序，将应填入(n)处的子句写在答卷纸的对应栏内。

【程序说明】

该程序定义了两个子函数 strsort 和 strmerge。它们分别实现了将一个字符串按字母顺序排序和将两个字符串合并排序，并删去相同字符。在主函数里，先输入两个字符串 s1 和 s2，然后调用 strsort 函数对它们分别排序，然后调用 strmerge 函数将 s1 和 s2 合并，将合并后的字符串赋给字符串 s3，最后输出字符串 s3。

【C 程序】

```
#include <stdio.h>
void strmerge(char *a,char *b,char *c)//将字符串 a,b 合并到字符串 c 中
{
  char t,*w;
  w=c;
  while(___(1)___)
  {
  //找到字符串 a,b 当前字符中较小的字符
    if(*a<*b)
    {
      t=*a;
      ___(2)___;
    }
    else if(*a>*b)
    {
      t=*b;
      ___(3)___;
    }
    else //字符串 a,b 当前字符相等
    {
      t=*a;
      a++;
      b++;
    }
```




```
    if(___(4)___)//开始，可直接赋值
      *w=t;
    else if(t!=*w)
    //如果 a、b 中较小的当前字符与 c 中当前字符不等，才赋值
      ___(5)___;
  }
  if(*a!='\0') //如果字符串 a 还没有结束，则将 a 的剩余部分赋给 c
  while(*a!='\0')
    if(*a!=*w)
    {
      *(++w)=*a;
      a++;
    }
    else
      ___(6)___;
  if(*b!='\0') //如果字符串 b 还没有结束，则将 b 的剩余部分赋给 c
  while(*b!='\0')
    if(*b!=*w)
    {
      *(++w)=*b;
      b++;
    }
    else
      b++;
      ___(7)___;
  }

void strsort(char *s)//将字符串 s 中的字符排序
{
  int i,j,n;
  char t,*w;
  w=s;
  for(n=0;*w!='\0';n++) //得到字符串长度 n
    w++;
  for(i=0;i<n-1;i++) //对字符串 s 进行排序，按字母先后顺序
    for(j=i+1;j<n;j++)
      if(___(8)___)
        {
          t=s[i];
          s[i]=s[j];
          ___(9)___;
        }
  }

void main()
{
  char s1[100],s2[100],s3[100];
  printf("\nPlease input the first string:");
  scanf("%s",s1);
  printf("\nPlease input the second string:");
```

```
    scanf("%s",s2);
    strsort(s1);//将字符串 s1 排序
    strsort(s2); //将字符串 s2 排序
    printf("%s\n",s1);
    printf("%s\n",s2);
    s3[0]='\0'; //字符串 s3 的第一个字符先置'\0'结束标志
    ___(10)___;//将 s1 和 s2 合并，按照字母顺序排列，
               //且要删去相同字符，存入 s3 中
    printf("%s",s3);
}
```

试题六(共 15 分)

阅读以下说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

现要编写一个画矩形的程序，目前有两个画图程序 DP1 和 DP2，DP1 用函数 draw_a_line(x1, y1, x2, y2)画一条直线，DP2 则用 drawline(x1, x2, y1, y2)画一条直线。当实例化矩形时，确定使用 DP1 还是 DP2。

为了适应变化，需要设计“不同类型的形状”和“不同类型的画图程序”，将抽象部分与实现部分分离，使它们可以独立地变化。这里，“抽象部分”对应“形状”，“实现部分”对应“画图”，与一般的接口(抽象方法)和具体实现不同。这种应用称为 Bridge(桥接)模式。图 7-33 显示了各个类间的关系。

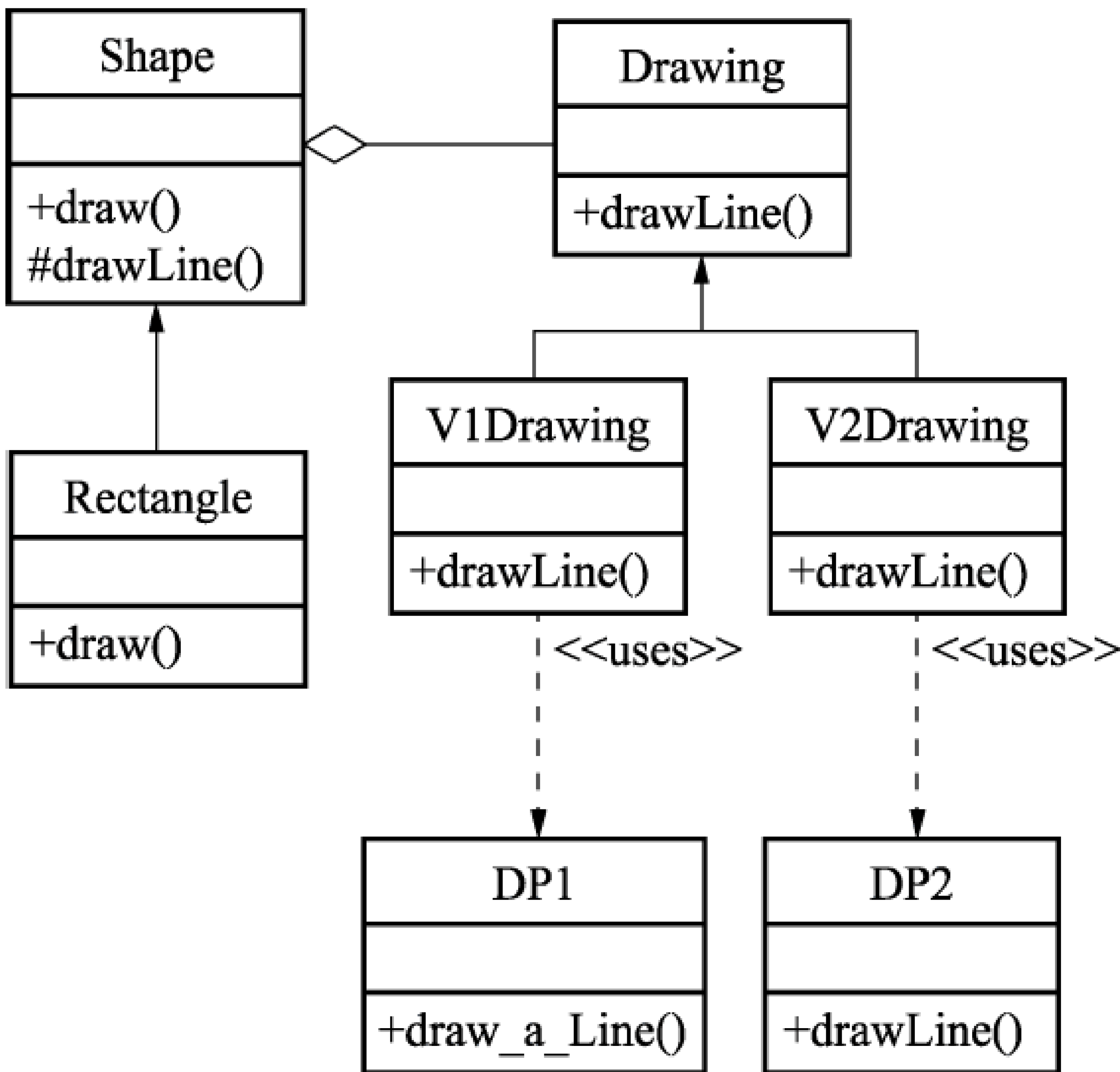
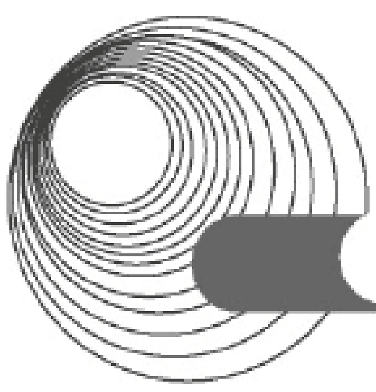


图 7-33　各个类间的关系

这样，系统始终只处理 3 个对象：Shape 对象、Drawing 对象、DP1 或 DP2 对象。以下是 C++语言实现，能够正确编译通过。

【C++程序】

```
class DP1{
public:
```




```
    static void draw_a_line(double x1, double y1,double x2, double y2){
    //省略具体实现
    }
};

class DP2{
public:
    static void drawline(double x1, double x2, double y1, double y2){
    //省略具体实现
    }
};

class Drawing{
public:
____(1)____void drawLine(double x1,double y1,double x2,double y2)=0;
};
class V1Drawing:public Drawing{
public:
    void drawLine(double x1, double y1, double x2, double y2){
        DP1::draw_a_line(x1, y1, x2, y2);
    }
};

class V2Drawing:public Drawing{
public:
    void drawLine(double x1, double y1, double x2, double y2){
        ____(2)____;
    }
};

class Shape{
private:
____(3)____ _dp;
public:
    Shape(Drawing *dp);
    virtual void draw() = 0;
    void drawLine(double x1, double y1, double x2, double y2);
};
Shape::Shape(Drawing *dp)
{
    _dp = dp;
}
void Shape::drawLine(double x1, double y1, double x2, double y2)
{   //画一条直线
    ____(4)____;
}

class Rectangle:public Shape{
private:
    double _x1, _y1, _x2, _y2;
public:
```

```
    Rectangle(Drawing *dp, double x1, double y1,
              double x2, double y2);
    void draw();
};
Rectangle::Rectangle(Drawing *dp, double x1, double y1, double x2, double y2)
                                  : ___(5)___
{
    _x1 = x1; _y1 = y1; _x2 = x2; _y2 = y2;
}
void Rectangle::draw()
{
//省略具体实现
}
```

试题七(共 15 分)

阅读以下函数说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

现要编写一个画矩形的程序，目前有两个画图程序 DP1 和 DP2，DP1 用函数 draw_a_line(x1, y1, x2, y2)画一条直线，DP2 则用函数 drawline(x1, x2, y1, y2)画一条直线。当实例化矩形时，确定使用 DP1 还是 DP2。

为了适应变化，需要设计“不同类型的形状”和“不同类型的画图程序”，将抽象部分与实现部分分离，使它们可以独立地变化。这里，“抽象部分”对应“形状”，“实现部分”对应“画图”，与一般的接口(抽象方法)和具体实现不同。这种应用称为 Bridge(桥接)模式。图 7-34 显示了各个类间的关系。

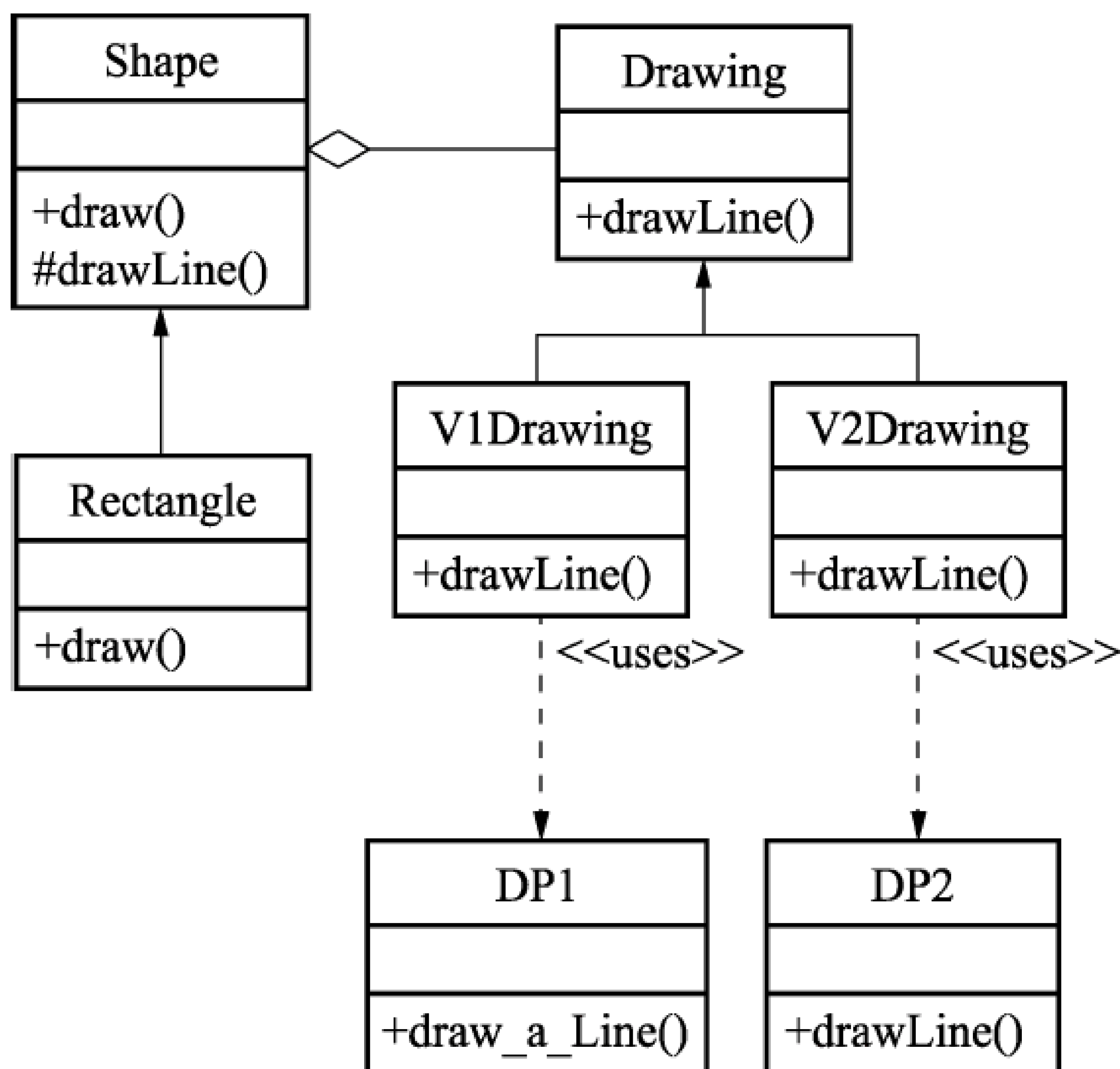


图 7-34　各个类间的关系

这样，系统始终只处理 3 个对象：Shape 对象、Drawing 对象、DP1 或 DP2 对象。以下是 Java 语言实现，能够正确编译通过。

【Java 程序】

```
//DP1.java 文件
public class DP1 {
    static public void draw_a_line(double x1, double y1,
                                double x2, double y2){
        //省略具体实现
    }
}

//DP2.java 文件
public class DP2 {
    static public void drawline(double x1, double y1,
                           double x2, double y2){
        //省略具体实现
    }
}

//Drawing.java 文件
__(1)__public class Drawing {
    abstract public void drawLine(double x1, double y1, double x2, double
y2);
}

//V1Drawing.java 文件
public class V1Drawing extends Drawing {
    public void drawLine(double x1, double y1, double x2, double y2){
        DP1.draw_a_line(x1, y1, x2, y2);
    }
}

//V2Drawing.java 文件
public class V2Drawing extends Drawing {
    public void drawLine(double x1, double y1,
                        double x2, double y2){ //画一条直线
        __(2)__;
    }
}

//Shape.java 文件
abstract public class Shape {
    abstract public void draw();
    private__(3)__dp;
    Shape(Drawing dp){
        _dp = dp;
    }
    protected void drawLine(double x1, double y1,
                        double x2, double y2){
        __(4)__;
    }
```

```
}

//Rectangle.java 文件
public class Rectangle extends Shape {
    private double _x1, _x2, _y1, _y2;
    public Rectangle(Drawing dp,
            double x1, double y1,
            double x2, double y2){
        __(5)__;
        _x1 = x1; _x2 = x2;
        _y1 = y1; _y2 = y2;
    }
    public void draw(){
    //省略具体实现
    }
}
```

7.1.5　样卷五

全国计算机技术与软件专业技术资格(水平)考试样卷五

试题一~试题四是必答题

试题一(15 分)

阅读以下说明和流程图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

下面给出的是某房产管理系统的一套分层数据流图，其功能描述如下。

(1) 系统随时根据住户送来的入住单更新住户基本信息文件。

(2) 每月初系统根据物业管理委员会提供的月附加费(如清洁费、保安费和大楼管理费等)表和房租调整表，计算每家住户的月租费(包括月附加费)，向住户发出交费通知单。住户交费时，系统输入交费凭证，核对后输出收据给住户。

(3) 系统定期向物业管理委员会提供住房分配表和交费情况表。

(4) 住户因分户或换房，在更新住户基本信息文件的同时，系统应立即对这些住户做月租费计算，以了结分户或换房前的房租。

系统顶层图如图 7-35 所示，0 层图如图 7-36 所示。加工 1 子图和加工 2 子图分别如图 7-37 和图 7-38 所示。

假定题中提供的顶层图是正确的，请回答下列问题。

【问题 1】(5 分)

指出哪张图中的哪些文件可不必画出。

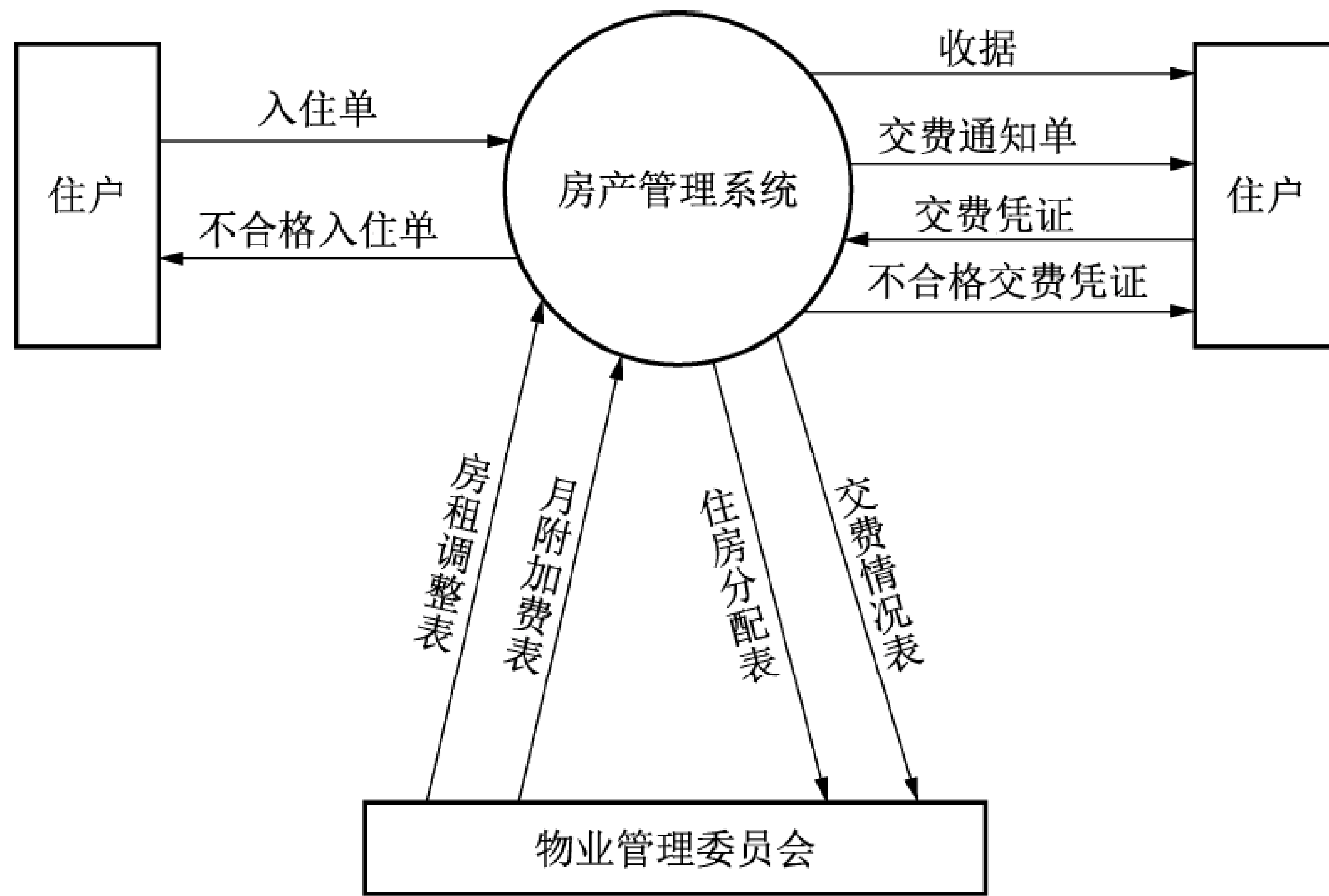


图 7-35　顶层图

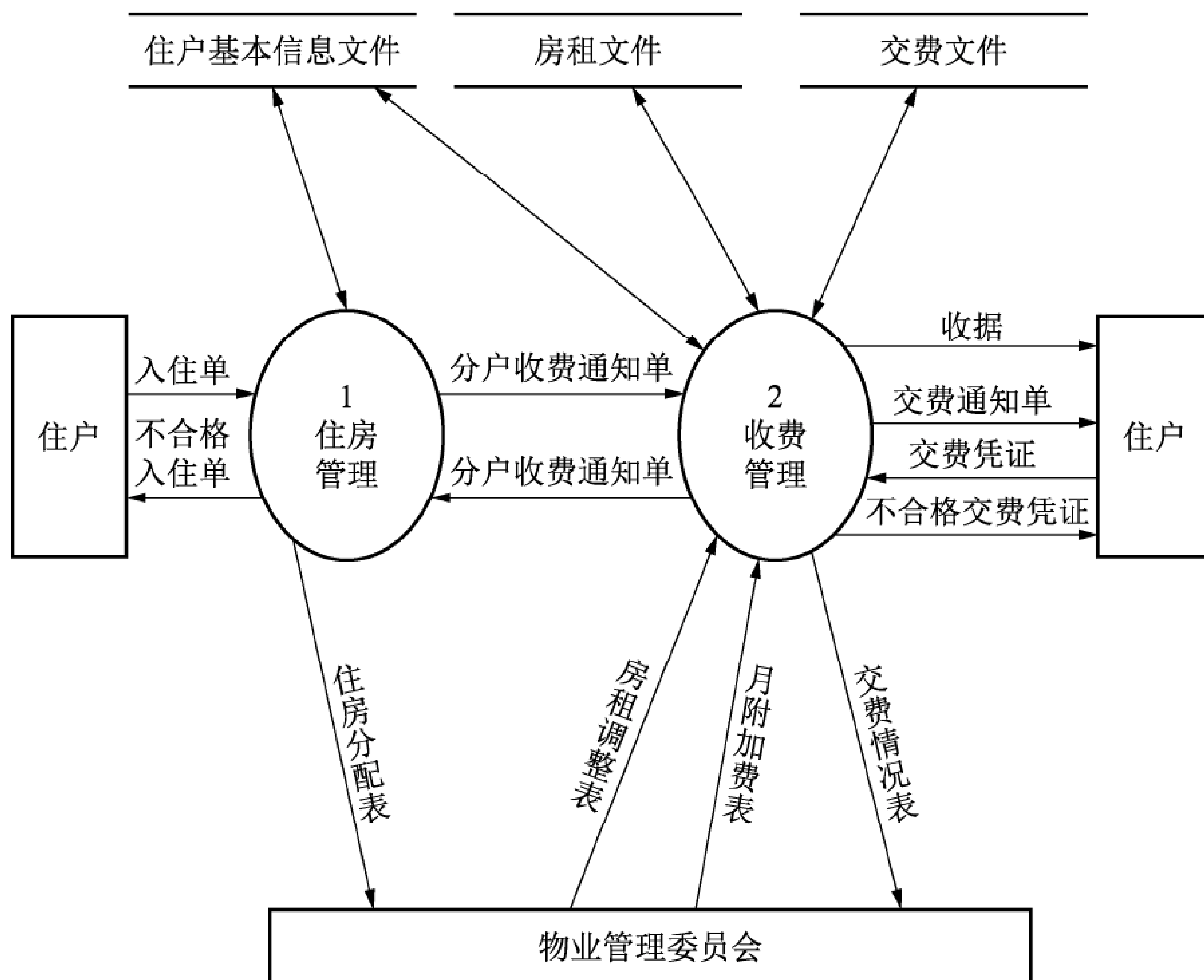


图 7-36　0 层图

【问题 2】(5 分)

指出在哪些图中遗漏了哪些数据流。回答时请用如下形式之一：

××图中遗漏了××加工(或文件)流向××加工(或文件)的××数据流；

××加工××遗漏了输入(或输出)数据流××。

【问题 3】(5 分)

指出图 7-38 中加工 2.3 能检查出哪些不合格交费凭证。



图 7-37　加工 1 子图



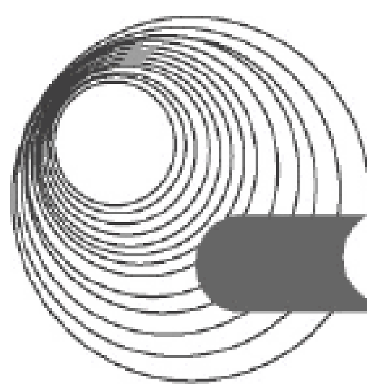
图 7-38　加工 2 子图

试题二(15 分)

阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

图书管理系统详细记录图书库存情况、读者信息及读者借阅记录(包括借书日期和还书日期)。



新书入库时要为该书编制图书卡片，包括分类目录号、图书流水号(要保证每本书都有唯一的流水号，即使同类图书也是如此)、书名、作者、内容摘要、价格和购书日期。同一个书名由于版次、作者等不同有可能存在多“种”图书，其间用“分类目录号”区分。

系统为每一位合法读者编制一个唯一的借书证号，读者需要提供姓名、单位。

一个读者最多可以同时借阅 5 本图书。借阅图书时，新添借阅记录，并将对应的“归还标记”字段置为“false”，表示“尚未归还”；归还图书时，将相应的“归还标记”字段置为“true”，表示“已经归还”。一本书可能供多位读者借阅，同一本书读者可以重复借阅。

图 7-39 所示为该系统的 E-R 图。

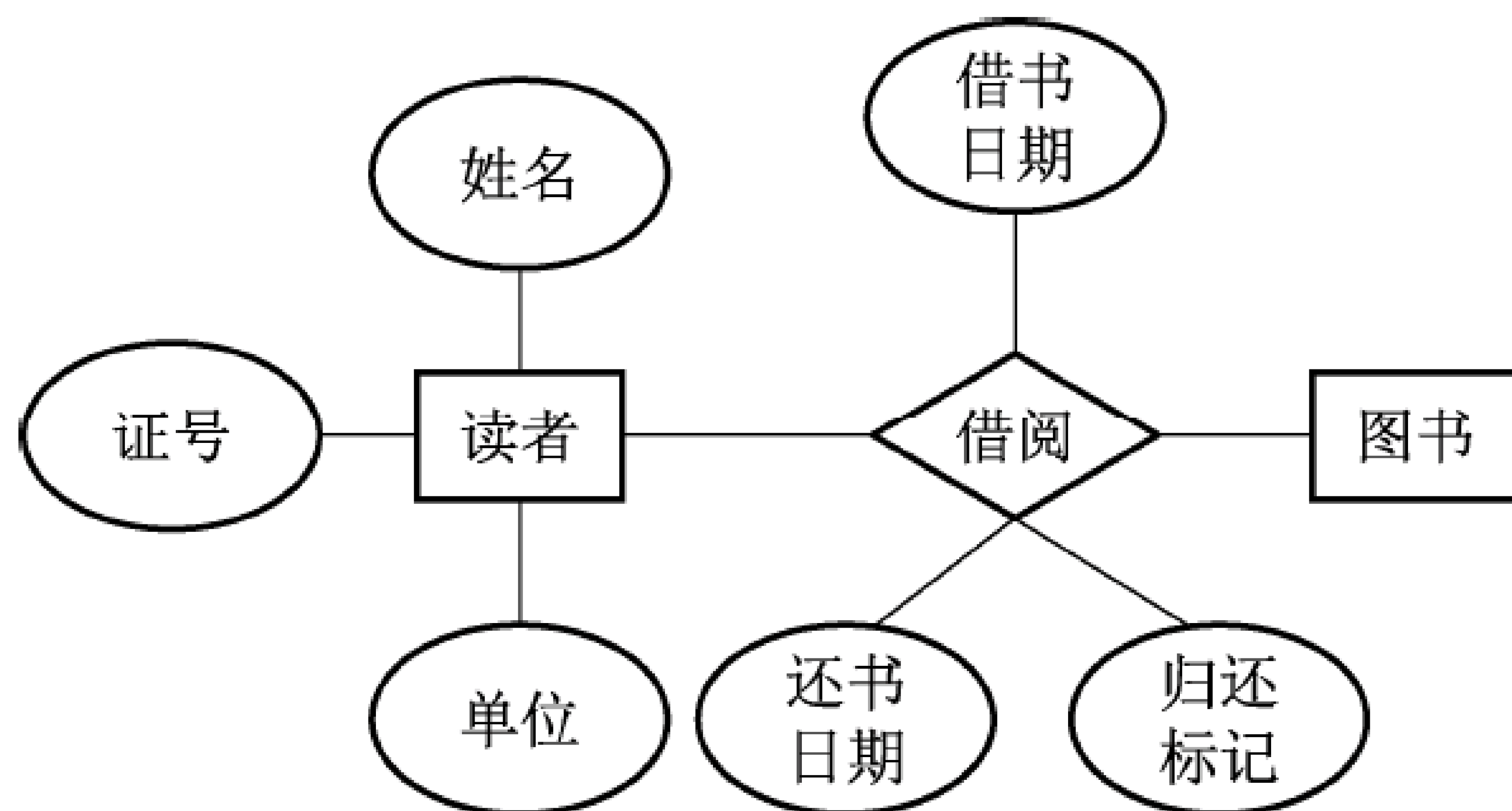


图 7-39　系统 E-R 图

【问题 1】(6 分)

实体间的联系有“一对一”“一对多”和“多对多”，指出“借阅”联系属于哪一种？“借阅”关系模式的外键是什么？有主键吗？为什么？

【问题 2】(5 分)

由于同一个分类目录号(同一种图书)有多个副本，若用表 Book(图书流水号，分类目录号，书名，作者，内容摘要，价格，购书日期)存储图书信息，则有很多的冗余信息，该如何分解使之满足 BCNF，并指出分解后的关系模式的主键。

【问题 3】(4 分)

设用表 Reader 存储读者信息，表 Book 存储图书信息，表 Borrow 存储借阅情况。

以下 SQL 语句用于“查询证号为 12345 的读者当前所借阅的图书书名(即尚未归还的图书)”，请补充完整。

```
SELECT 书名 FROM Book WHERE 流水号 __(1)__
        (SELECT 流水号 FROM __(2)__ WHERE 证号 = "12345" AND __(3)__)
```

以下 SQL 语句用于“查询书名包含‘软件设计师’的图书情况”，请补充完整。

```
SELECT * FROM Book WHERE 书名 __(4)__ "%软件设计师%"
```

试题三(15 分)

阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

某大型旅店为了便于管理，欲开发一个客房管理系统。希望实现客房预订、入住登记、账务结算、退房，以及将服务项目记入客人账单等功能。

旅客包括散客和团体，散客预订或入住时需要提供姓名、性别、身份证和联系电话，团体则提供团体名称、负责人的姓名、性别、身份证和联系电话，以及团体人数。对于散客还要提供换房服务。

旅店还提供了很多服务项目，比如早餐。对每一位入住客人，服务列表记录了住宿期间的各项服务，包括服务类型、日期、数量等。当然，客人也可以不要任何服务。

旅店的客房有一个唯一的房间号，分为不同的类别、不同的房间床位数和不同的价格。

为了有效的管理，系统需要记录每天的客房状态。客房的状态有空闲、占用、已预订和维修。

- 客人入住后，客房处于占用状态。
- 客人退房后，客房处于空闲状态。
- 客人预订后，客房处于已预订状态。
- 预订客人入住后，客房处于占用状态。
- 预订客人取消预订后，客房处于空闲状态。
- 需要维修时，客房处于维修状态。
- 维修完成后，客房处于空闲状态。

该系统采用面向对象方法开发，系统中的类及类之间的关系用 UML 类图表示。图 7-40 是该系统的类图的一部分；图 7-41 描述了客房状态的转变情况。

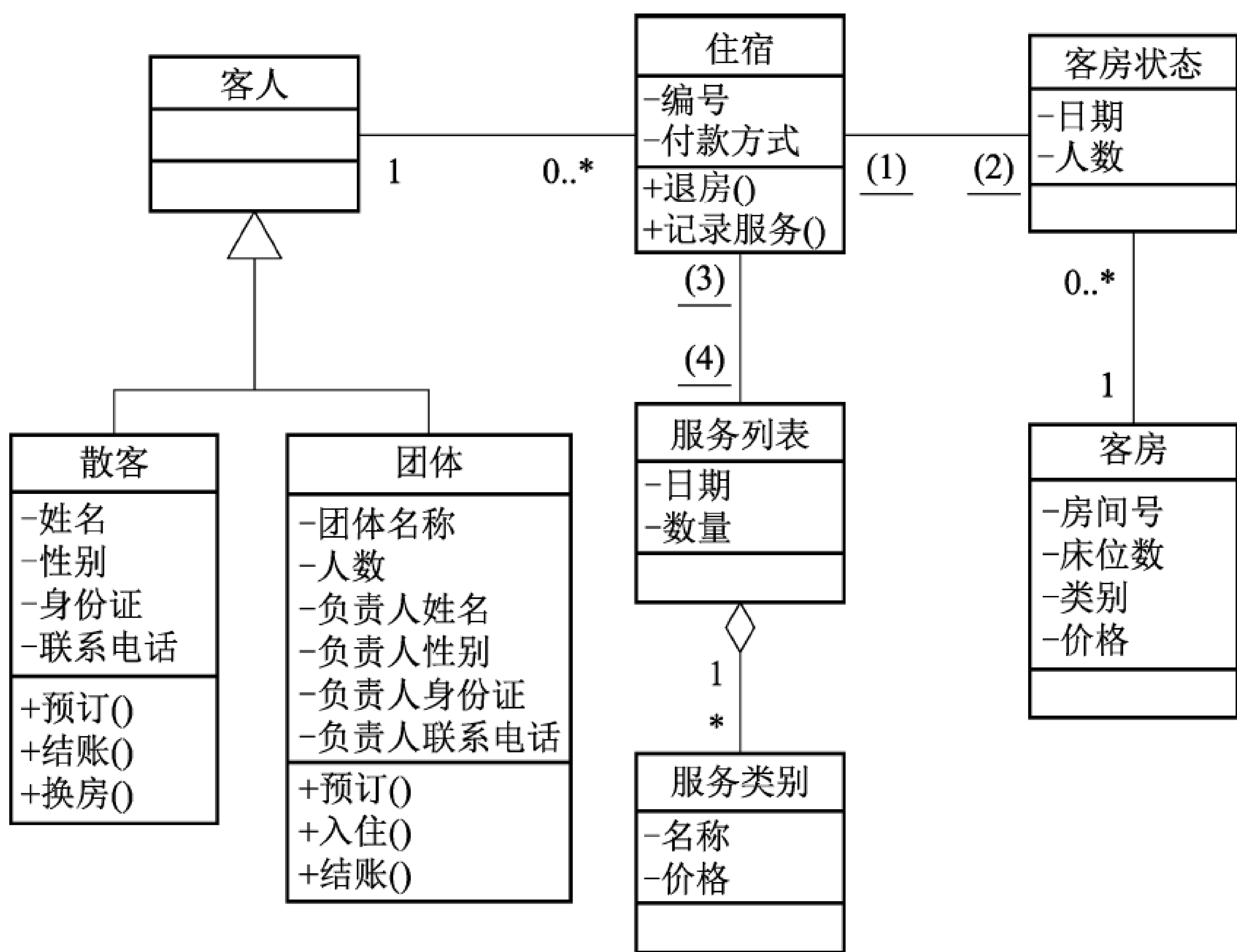


图 7-40　类图的一部分

【问题 1】(5 分)

请用如图 7-40 所示的属性和方法的名称给出“客人”类的属性和方法。(注意：“团体”类中的负责人姓名等与散客的对应属性含义相同，不必区分。)

【问题 2】(6 分)

在 UML 中，重复度(Multiplicity)定义了某个类的一个实例可以与另一个类的多少个实

例相关联。通常把它写成一个表示取值范围的表达式或者一个具体的值。例如，图 7-40 中的类“客人”和“住宿”，“客人”类端的“1”表示一个“住宿”类的实例只能与一个“客人”类的实例相关联；“住宿”类端的“0..*”表示一个“住宿”类的实例可以与 0 个或多个“客人”类的实例相关。请指出图 7-40 中(1)~(4)处的重复度分别为多少。

【问题 3】(4 分)

根据题意，请指出图 7-41 中状态 A、B 分别是什么状态，事件 C、D 分别是什么事件。

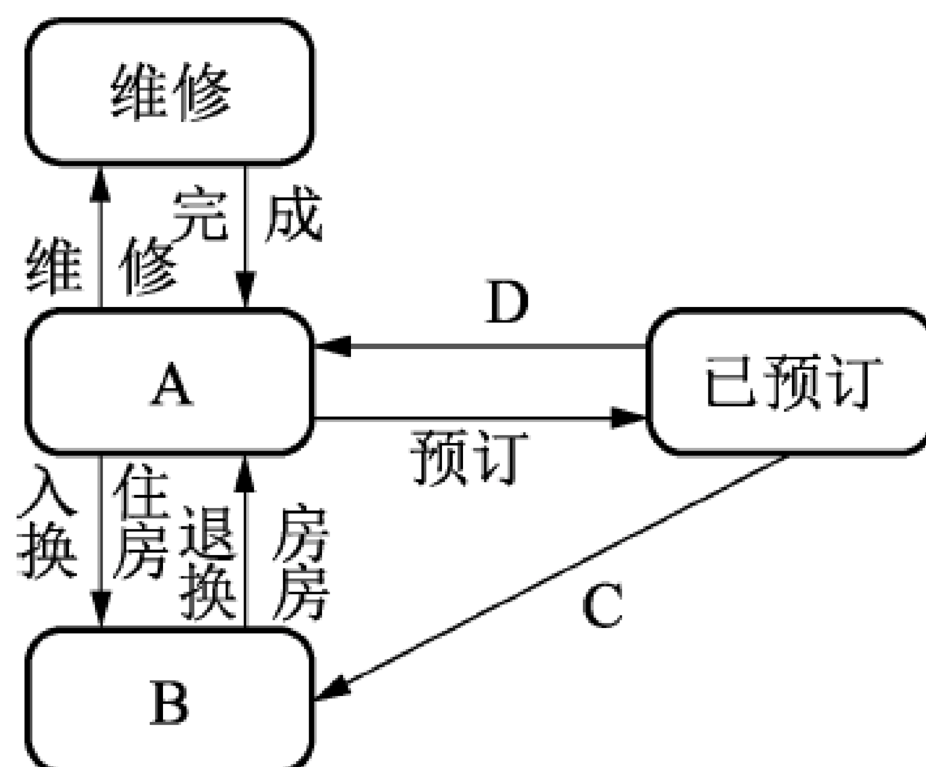


图 7-41　客房状态的转变情况

试题四(15 分)

分析图 7-42 所示的流程图并回答问题。

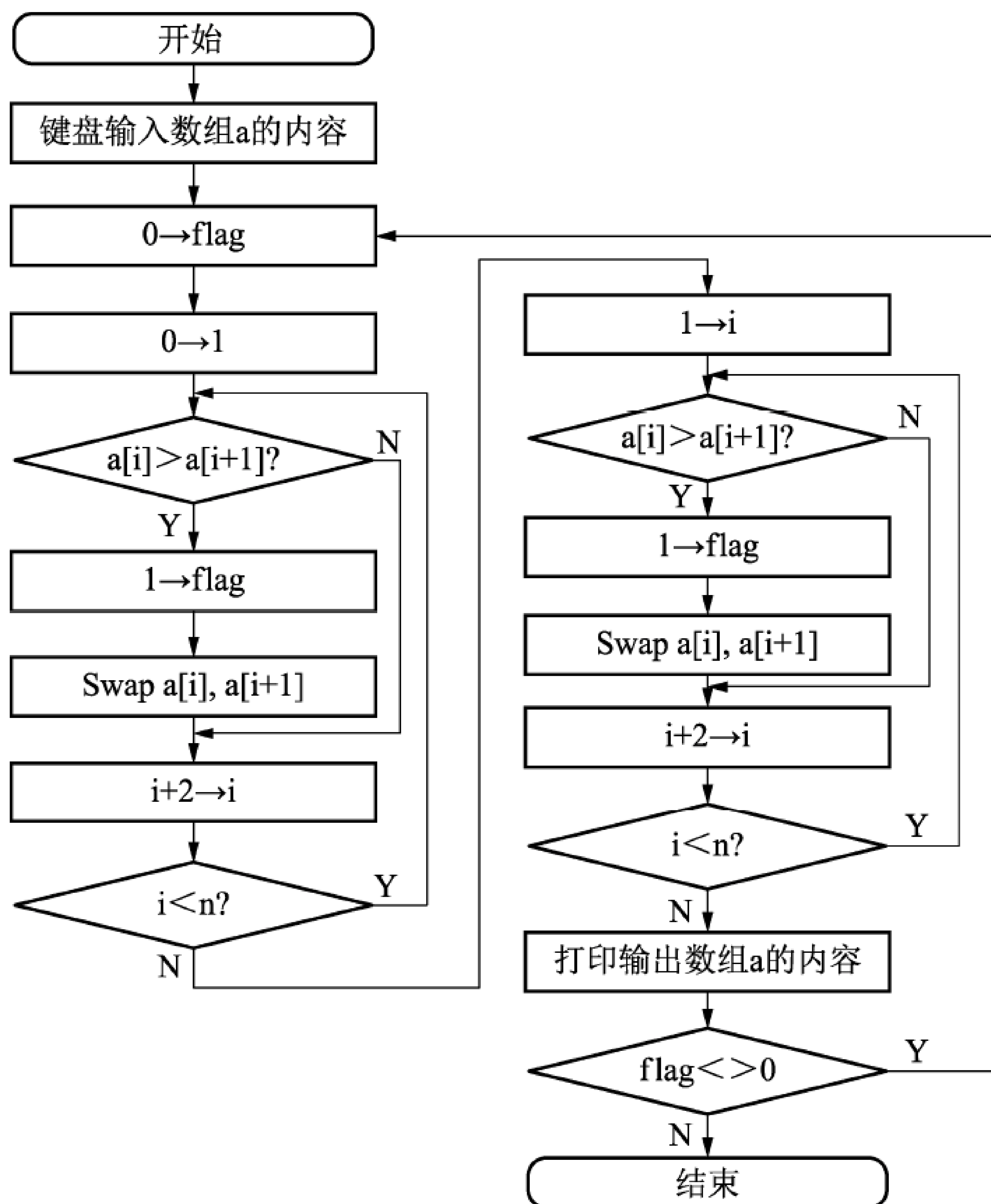


图 7-42　流程图

【问题 1】(5 分)

上面程序的功能是什么？

【问题 2】(10 分)

若数组 a 输入的数为{10,8,15,2,7,13,4}，请写出 a 的前 3 趟输出结果。

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(15 分，每空 3 分)

阅读以下说明和程序，将应填入(n)处的子句写在答卷纸的对应栏内。

【说明】

下面程序为堆排序程序，其中函数 adjust(i,n)是把以 R[i]($1 \leqslant i \leqslant \lfloor i/2 \rfloor$)为根的二叉树调整成堆的函数。假定 R[i]的左、右子树已经是堆，程序中的 r 是在主函数中说明的结构数组，它含有要排序的 n 个记录。

【程序】

```
void adjust(i,n)
int i,n;
{
    int k,j;
    element extr;
    extr=r[i];
    k=i;
    j=2*i;
    while(j<=n)
{
    if((j<n)&&(r[j].key<r[j+1].key))
    ___(1)___;
    if(extr.key<r[j].key)
    {
        r[k]=r[j];
        k=j;
    ___(2)___;
    }
    else
    ___(3)___;
}
r[k]=extr;
}
```

让 i 从$\lfloor i/2 \rfloor$逐步减到 1，反复调用函数 adjust，便完成建立初始堆的过程，heapsort 程序如下。

```
void heapsort(r,n)
list r;
int n;
```

```
{
    int i,l;
    element extr;
    for (i=n/2;i>=1;--i)
    ___(4)___; /*建立初始堆*/
    for(k=n;k>=2;k--)
    {
        extr=r[1];
        r[1]=r[k];
        r[k]=extr;
        ___(5)___;
    }
}
```

试题六(共 15 分)

阅读以下说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

以下 C++程序的功能是计算三角形、矩形和正方形的面积并输出。程序由 4 个类组成：类 Triangle、Rectangle 和 Square 分别表示三角形、矩形和正方形；抽象类 Figure 提供了一个纯虚函数 getArea()，作为计算上述 3 种图形面积的通用接口。

【C++程序】

```
#include <iostream>
#include <cmath>
using namespace std;

class Figure{
public:
    virtual double getArea() = 0;//纯虚函数
};

class Rectangle : _(1)_{
protected:
    double height;
    double width;
public:
    Rectangle(){}
    Rectangle(double height, double width){
        This->height = height;
        this->width = width;
    }
    double getArea(){
        return _(2)_;
    }
};

class Square : _(3)_{
public:
```

```
    Square(double width){
        (4);
    }
};

class Triangle: (5) {
private:
    double la, lb, lc;
public:
    Triangle(double la, double lb, double lc){
        this->la = la; this->lb = lb; this->lc = lc;
    }
    double getArea(){
        double s = (la+lb+lc) / 2.0;
        return sqrt(s*(s-la)*(s-lb)*(s-lc));
    }
};

int main()
{
    Figure *figures[3]={new Triangle(2, 3, 3), new Rectangle(5, 8), new
Square(5)};
    for(int i = 0; i < 3; i++){
        cout<<"figures["<<i<<"]area = "<<(figures[i]) ->getArea()<<endl;
    }
    return 0;
}
```

试题七(共 15 分)

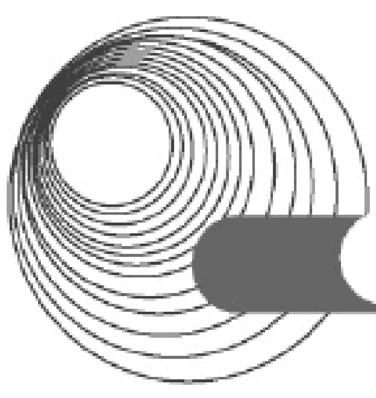
阅读以下函数说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

下面的程序先构造 Point 类，再顺序构造 Ball 类。由于在 Ball 类中不能直接存取 Point 类中的 xCoordinate 及 yCoordinate 属性值，因此 Ball 类中的 toString 方法调用 Point 类中的 toString 方法输出中心点的值。在 MovingBall 类的 toString 方法中，super.toString 调用父类 Ball 的 toString 方法输出 Ball 类中声明的属性值。

【Java 程序】

```
//Point.java 文件
public class Point{
    private double xCoordinate;
    private double yCoordinate;
    public Point(){}
    public Point(double x, double y){
        xCoordinate = x;
        yCoordinate = y;
    }
    public String toString(){
```




```
        return "("+Double.toString(xCoordinate)+","
        +Double.toString(yCoordinate)+")";
    }
    //other methods
}
//Ball.java 文件
public class Ball{
    private  (1) ;//中心点
    private double radius;//半径
    private String color;//颜色
    public Ball(){}
    public Ball(double xValue, double yValue, double r){
        //具有中心点及其半径的构造方法
        center =  (2) ;//调用类 Point 中的构造方法
        radius = r;
    }
    public Ball(double xValue, double yValue, double r, String c){
        //具有中心点、半径和颜色的构造方法
         (3) ;//调用 3 个参数的构造方法
        color = c;
    }
    public String toString(){
        return "A ball with center "+center.toString()
        +", radius "+Double.toString(radius)+", color "+color;
    }
    //other methods
}
class MovingBall  (4) {
    private double speed;
    public MovingBall(){}
    public MovingBall(double xValue, double yValue, double r, String c, double
s){
         (5) ;//调用父类 Ball 中具有 4 个参数的构造方法
        speed = s;
    }
    public String toString(){
        return super.toString()+", speed "+Double.toString(speed);
    }
    //other methods
}
public class test {
    public static void main(String args[]){
        MovingBall mb = new MovingBall(10, 20, 40, "green", 25);
        System.out.println(mb);
    }
}
```

7.1.6　样卷六

全国计算机技术与软件专业技术资格(水平)考试样卷六

试题一~试题四是必答题

试题一(共 15 分)

阅读下列说明和数据流图图 7-43~图 7-45，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

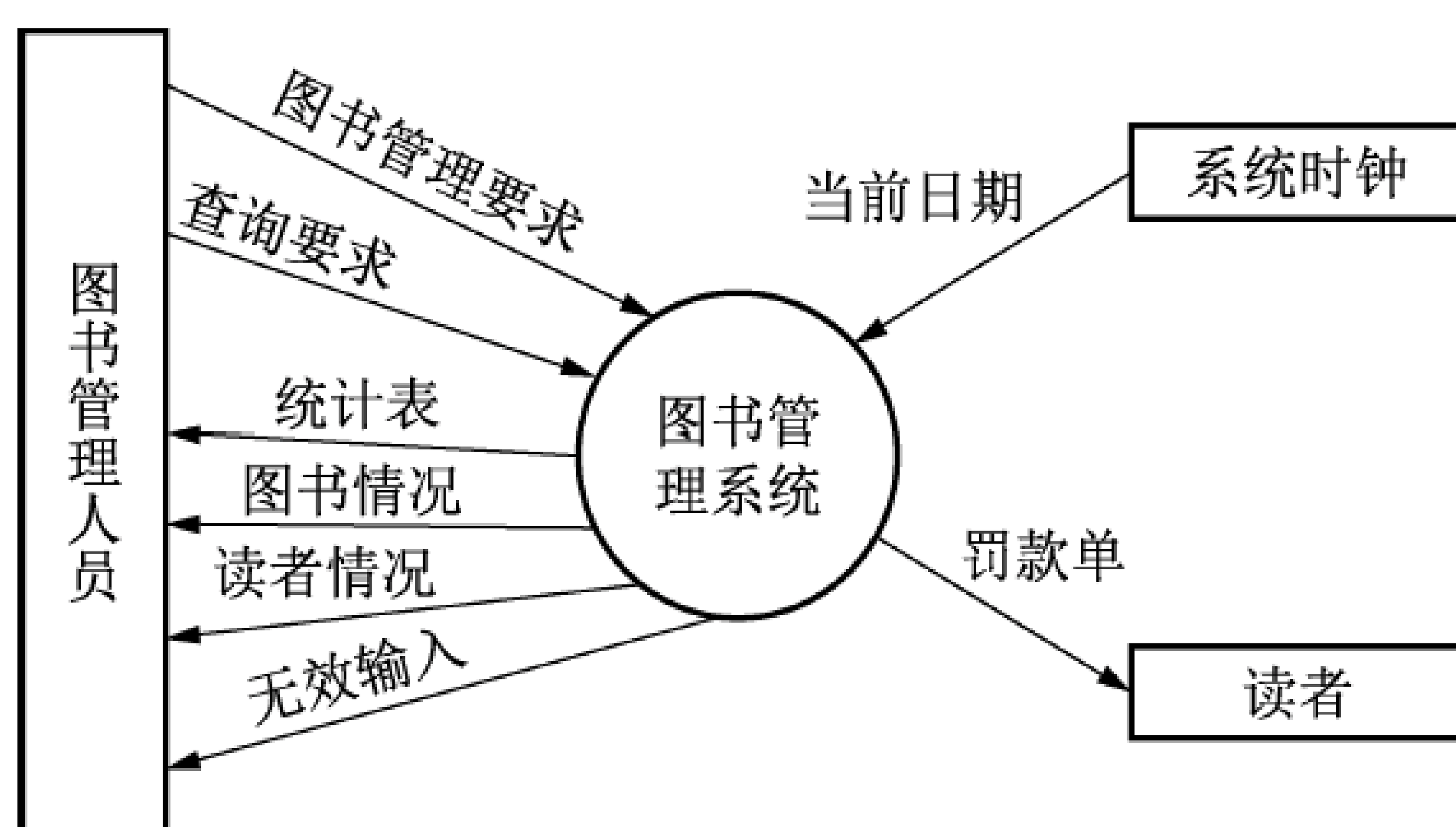


图 7-43　顶层数据流图

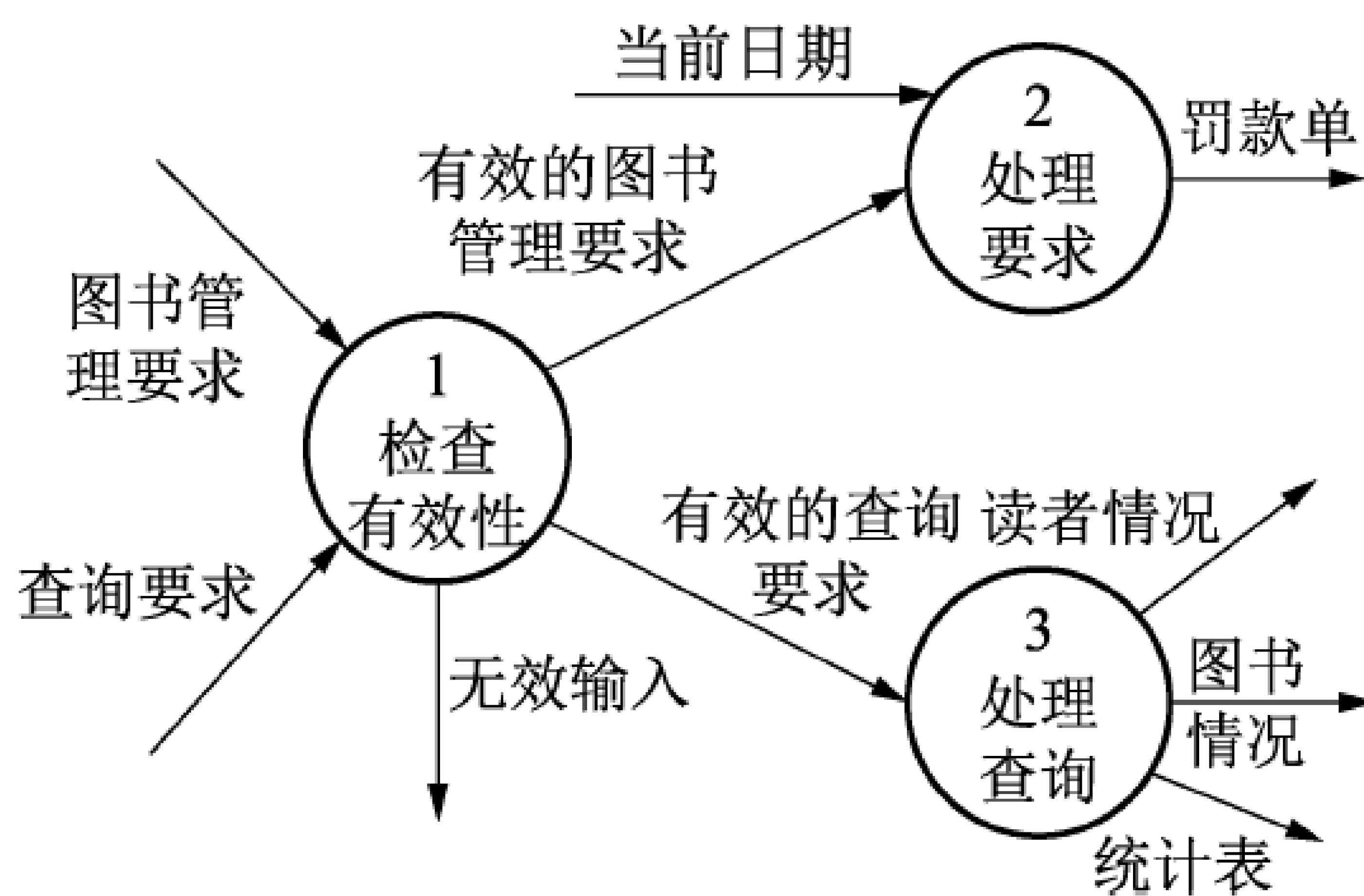


图 7-44　0 层数据流图

【说明】

图书管理系统旨在用计算机对图书进行管理，包括图书的购入、借阅、归还及注销。管理人员可以查询某位读者、某种图书的借阅情况，还可以对当前图书借阅情况进行一些统计，给出统计表格，以便掌握图书的流通情况。

系统要实现以下 5 个方面的功能：购入新书、读者借书、读者还书、图书注销及流通查询。

(1) 购入新书：需要为新书编制图书卡片，包括分类目录号、图书流水号(要保证每本书都有唯一的流水号，即使同类图书也是如此)、书名、作者、内容摘要、价格和购书日期等信息，写入图书目录文件中。

(2) 读者借书：填写借书单，包括读者号、欲借图书分类目录号，系统首先检查该读者号是否有效，若无效，则拒绝借书，否则进一步检查该读者所借图书是否超过最大限制数，若已达到最大借阅数，则拒绝借书，否则读者可以借出该书，登记图书分类目录号、图书流水号、读者号和借阅日期等，写回到借书文件中。

(3) 读者还书：根据图书流水号，从借书文件中读出和该图书相关的借阅记录，标明还书日期，再写回借书文件中；如果图书逾期未还，则处以相应罚款。

(4) 图书注销：将一些过时或无保留价值的图书注销，从图书文件中删除相关记录。

(5) 流通查询：管理员可以对图书流通情况进行查询，包括某位读者、某种图书和全局图书，给出流通情况统计表。

以下是经分析得到的数据流图及部分数据字典，有些地方有待填充，假定顶层数据流图是正确的。图 7-43 是顶层数据流图；图 7-44 是 0 层数据流图；图 7-45 是 1 层数据流图。

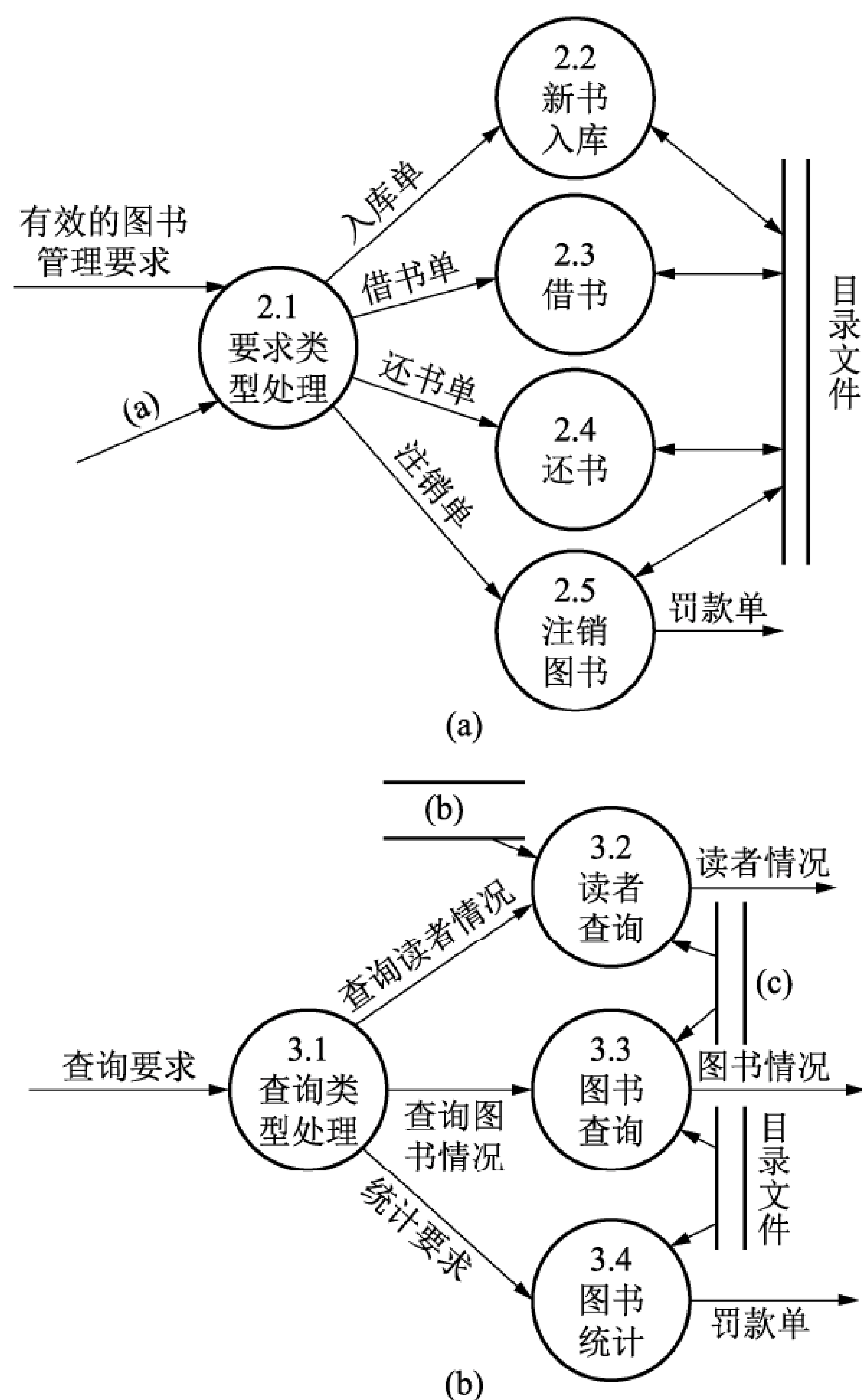


图 7-45　1 层数据流图

【数据字典】

1) 数据流条目

图书管理要求=[入库单|借书单|还书单|注销单]；

入库单=分类目录号+数量+书名+作者+内容摘要+价格+购书日期；

借书单=读者号+ (d) +借阅日期；

还书单= (e) +还书日期。

2) 文件说明

文件名：目录文件。

组成：{分类目录号+书名+作者+内容摘要+价格+入库日期+总数+库存数+ (f) }。

【问题 1】(4 分)

根据题意，指出图 7-45(a)中缺失的数据流(a)的名称，并指出该数据流的起点。

【问题 2】(4 分)

将下述文件正确填充在图 7-45(b)中(b)、(c)处：读者文件、借书文件。

【问题 3】(7 分)

根据题意，补充数据字典中(d)、(e)、(f)处的空缺。

试题二(共 15 分)

阅读下列说明和 E-R 图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

有个关于运动会的管理系统，在该系统中，委员会为每一个参赛的运动员赋予一个唯一的编号“运动员号”，同时记录姓名、性别、年龄和队名，其中姓名和队名必须填写。

一个运动员属于且只属于一个队，一个运动员可以参赛多个项目。运动员参加比赛取得一个成绩，相应有一个积分：第一名积分 6 分，第二名积分 4 分，第三名积分 2 分，其他的没有积分。一个队的总积分是该队所有队员的积分之和。

图 7-46 是该系统的 E-R 图。图中的实体和属性同时给出了中、英文两种名字，回答问题时只需写出英文名即可。

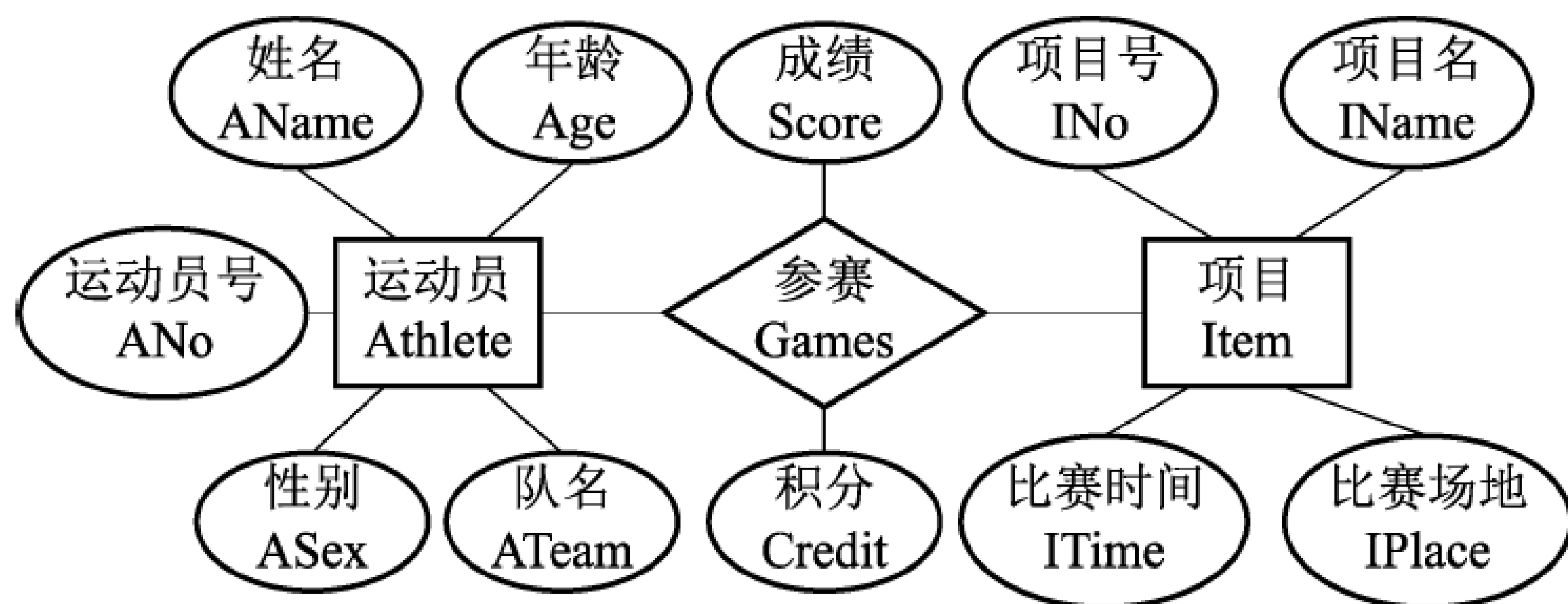


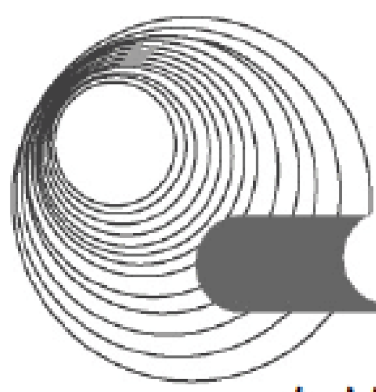
图 7-46　系统 E-R 图

【问题 1】(6 分)

根据 E-R 图中给出的词汇，按照“有关模式名(属性, 属性, …)”的格式，将此 E-R 图转换为 3 个关系模式，指出每个关系模式中的主键，其中模式名根据需要取实体名或联系名。

【问题 2】(3 分)

创建 Athlete 表时，ANo 使用 CHAR(6)并且唯一，AName 使用 CHAR(20)，ASex 使用 CHAR(1)，ATeam 使用 CHAR(20)。请在下列用于创建表 Athlete 的 SQL 语句空缺处填入正



确的内容。

```
CREATE TABLE Athlete(ANo CHAR(6) NOT NULL,
                     AName CHAR(20),
                     ASex CHAR(1),
                     ATeam CHAR(20) NOT NULL,
                     __(1)__);
```

【问题 3】(6 分)

假定 Games 表存储参赛情况，以下的 SQL 语句是委员会用于查询“队名为‘China’的各个运动员各自取得的总积分”的不完整语句，请在空缺处填入正确的内容。

```
SELECT __(2)__
       FROM Games
        WHERE ANo __(3)__
        (SELECT ANo
           FROM __(4)__
           WHERE ATeam = "China")
           GROUP BY ANo;
```

试题三(共 15 分)

阅读下列说明和图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

C 市刚开通了地铁线，为方便乘客，计划开发自动售票系统。

该公司在每一个地铁站放置了多台自动售票机，每一台售票机有唯一编号，售票记录统一汇总主机。自动售票机只发售从该站起始的各种地铁票，因此乘客只需输入目的站，起始站默认为该站，售票机给出从该站到达目的站的单程票。打印地铁票时为其编一个唯一的流水号，并同时打印自动售票机的编号及票价。

售票机的状态变化如下：“空闲”状态时，显示地铁线路图，等待乘客输入目的站；当乘客输入目的站后，转入“目的站确认/票数输入”状态，同时给出票价，此时若目的站有误，可返回到空闲状态重新输入，否则，输入票数；乘客输入票数后，转入“票数确认/付款”状态，同样此时若票数有误，可返回到上一状态重新输入，否则，投入钱币付款；当付款金额足够时，转入“出票/找零”状态(有必要时进行找零)；然后转入“空闲”状态等待输入目的站状态。

该系统采用面向对象方法开发，系统中的类及类之间的关系用 UML 类图表示。图 7-47 是该系统类图的一部分；图 7-48 描述了自动售票机的状态转换图。

【问题 1】(2 分)

根据题意，给出“自动售票机”类的主要属性。

【问题 2】(5 分)

根据题意，给出“地铁票”类的主要属性。

【问题 3】(8 分)

根据题中所述术语，指出图 7-48 中状态 1~状态 4 分别是什么？

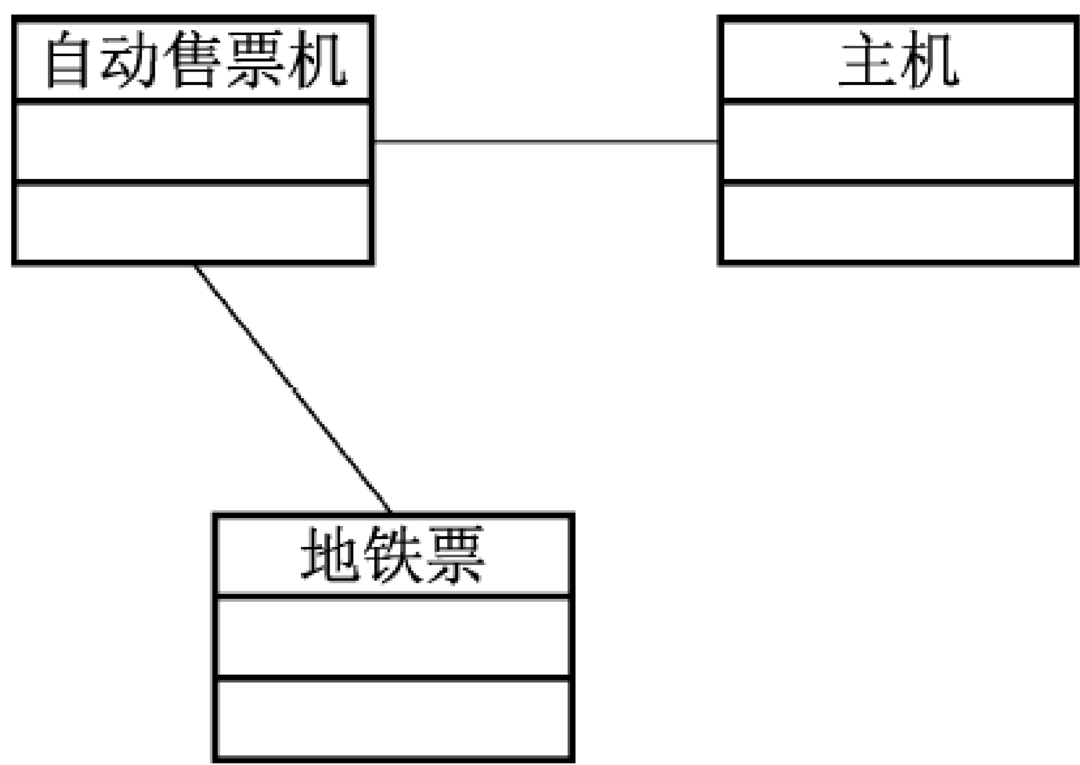


图 7-47　类图的一部分

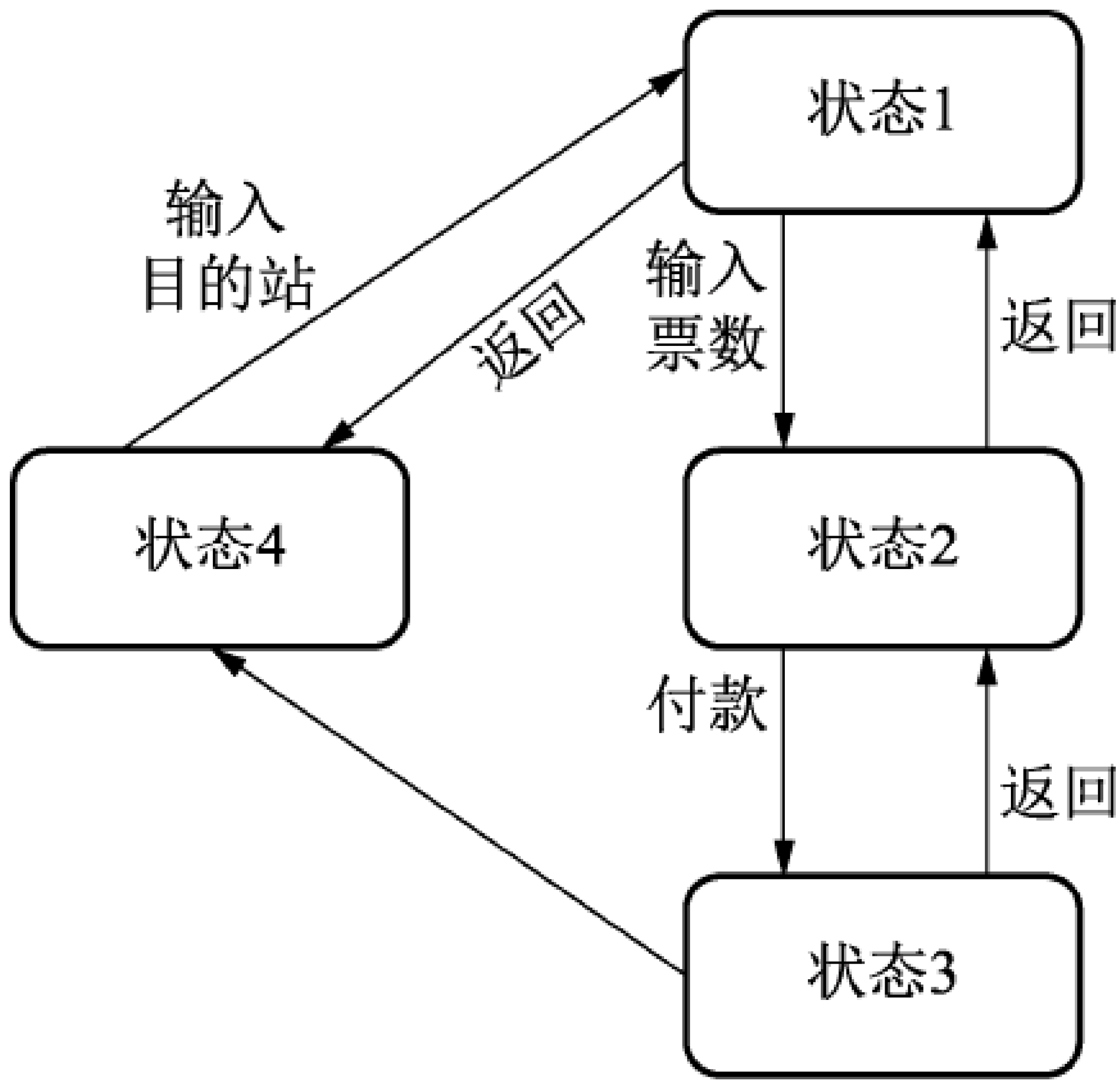


图 7-48　状态转换图

试题四(共 15 分)

阅读下列程序说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【程序说明】

设 M 叉树采用列表法表示，即每棵子树对应一个列表，列表的结构为：子树根节点的值部分(设为一个字符)和用“()”括起来的各子树的列表(如有子树的话)，各子列表间用“,”分隔。例如，如图 7-49 所示的三叉树可用列表 a(b(c, d), e, f(g, h, i))表示。

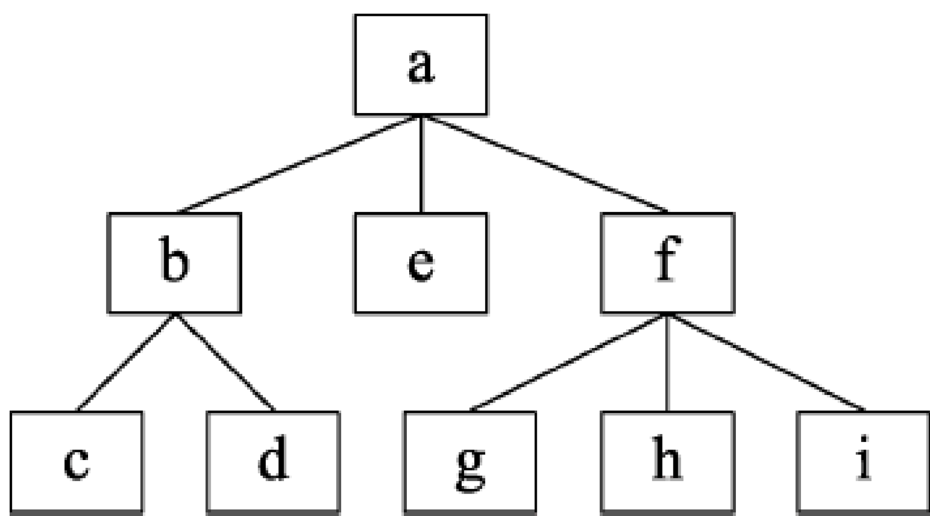


图 7-49　三叉树

本程序输入列表，生成一棵 M 叉树，并由 M 叉树输出列表。假定输入无错误。

【C 程序】

```
#include <stdio.h>
#include <stdlib.h>
#define M 3
typedef struct node{
    char val;
    struct node *subTree[M];
}NODE;
char buf[255], *str = buf;
NODE *d = NULL;
NODE *makeTree()  /*由列表生成 M 叉树*/
{
    int k; NODE *s ;
    s = (1);
    s->val = *str++;
    for(k = 0; k < M; k++)s->subTree[k] = NULL;
```

```
    if(*str == '('){
        k = 0;
        do{
            str++;
            s->subTree[k] = (2) ;
            if(*str == ')'){
                str++;
                break;
            }
            k = k+1;
        }while( (3) );
    }
    return s;
}
void walkTree(NODE *t) /*由 M 叉树输出列表*/
{
    int i ;
    if(t != NULL){
        (4) ;
        if(t->subTree[0] == NULL)return;
        putchar('(');
        for(i = 0; i < M; i++){
            (5) ;
            if(i != M-1 && t->subTree[i+1] != NULL)
                putchar(', ');
        }
        putchar(')') ;
    }
}
void main()
{
    printf("Enter exp:");
    scanf("%s", str);
    d = makeTree();
    walkTree(d);
    putchar('\n');
}
```

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(共 15 分)

阅读下列函数说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【函数说明】

若要在 N 个城市之间建立通信网络，只需要 N−1 条线路即可。如何以最低的经济代价

建设这个网络，是一个网的最小生成树的问题。现要在 8 个城市间建立通信网络，其拓扑结构如图 7-50 所示，边表示城市间的通信线路，边上标示的是建立该线路的代价。

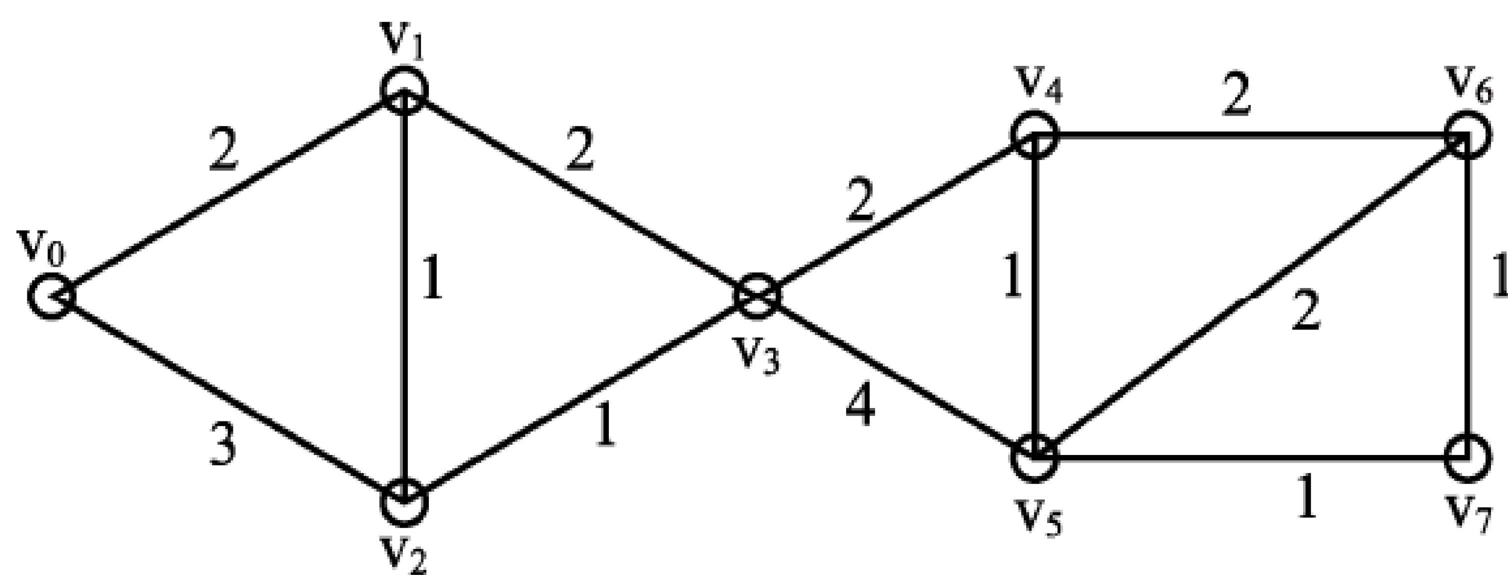


图 7-50　通信网络的拓扑结构

无向图用邻接矩阵存储，元素的值为对应的权值。考虑到邻接矩阵是对称的且对角线上元素均为 0，故压缩存储，只存储三角元素(不包括对角线)。

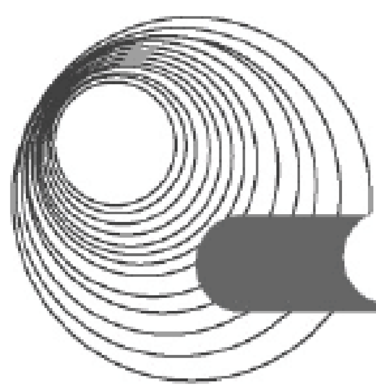
先用 Prim 算法生成网络的最小生成树。由网络 G = (V,E) 构造最小生成树 T = (U,TE) 的 Prim 算法的基本思想是：首先从集合 V 中任取一顶点放入集合 U 中，然后将一个顶点放在集合 U 里，另一个顶点放在集合 V-U 里的边中，找出权值最小的边(u, v)，将边加入 TE，并将顶点 v 加入集合 U，重复上述操作直到 U=V 为止。

函数中使用的预定义符号如下。

```
#define MAX 32768                          /*无穷大权，表示顶点间不连通*/
#define MAXVEX 30                          /*图中顶点数目的最大值*/
typedef struct{
    int StartVex, StopVex;                 /*边的起点和终点*/
    float weight;                          /*边的权*/
}Edge;
typedef struct{
    char vexs[MAXVEX];                     /*顶点信息*/
    float arcs[MAXVEX*(MAXVEX-1)/2];       /*邻接矩阵信息，压缩存储*/
    int n;                                 /*图的顶点个数*/
}Graph;
```

【函数 C 代码】

```
void PrimMST(Graph *pGraph, Edge mst[])
{
    int i, j, k, min, vx, vy;
    float weight, minWeight;
    Edge edge;
    for(i = 0; i < pGraph->n-1; i++){
        mst[i].StartVex = 0;
        mst[i].StopVex = i + 1;
        mst[i].weight = pGraph->arcs[i];
    }
    for(i = 0; i < (1); i++){/*共 n-1 条边*/
        minWeight = (float)MAX;
        min = i;
        /*从所有边(vx, vy)中选出最短的边*/
        for(j = i; j < pGraph->n-1; j++){
```




```
            if(mst[j].weight < minWeight){
                minWeight =  (2) ;
                min = j;
            }
        }
        /*mst[min]是最短的边(vx, vy),将mst[min]加入最小生成树*/
        edge = mst[min];
        mst[min] = mst[i];
        mst[i] = edge;
        vx =  (3) ;/*vx 为刚加入最小生成树的顶点下标*/
        /*调整mst[i+1]到mst[n-1]*/
        for(j = i+1; j < pGraph->n-1; j++){
            vy = mst[j].StopVex;
            if( (4) ){/*计算(vx, vy)对应的边在压缩矩阵中的下标*/
                k = pGraph->n*vy-vy*(vy+1)/2+vx-vy-1;
            }else{
                k = pGraph->n*vx-vx*(vx+1)/2+vy-vx-1;
            }
            weight =  (5) ;
            if(weight < mst[j].weight){
                mst[j].weight = weight;
                mst[j].StartVex = vx;
            }
        }
    }
}
```

试题六(共 15 分)

阅读以下说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

现有一个显示系统，要显示的图形有线 Line、矩形 Square，抽象出一个 Shape 类(接口)，有方法显示 display()。

需要新增图形 Circle，又已知有类 XXCircle 实现了所需要实现的功能：显示 displayIt()。为了继承自 Shape 以提供统一接口，又不希望从头开发代码，希望使用 XXCircle。这样将 XXCircle 作为 Circle 的一个属性，即 Circle 的对象包含一个 XXCircle 对象。当一个 Circle 对象被实例化时，它必须实例化一个相应的 XXCircle 对象；Circle 对象收到的做任何事的请求都将转发给这个 XXCircle 对象。通过这种被称为 Adapter 的模式，Circle 对象就可以通过“让 XXCircle 做实际工作”来表现自己的行为了。图 7-51 显示了各个类间的关系。以下是 C++语言实现，能够正确编译通过。

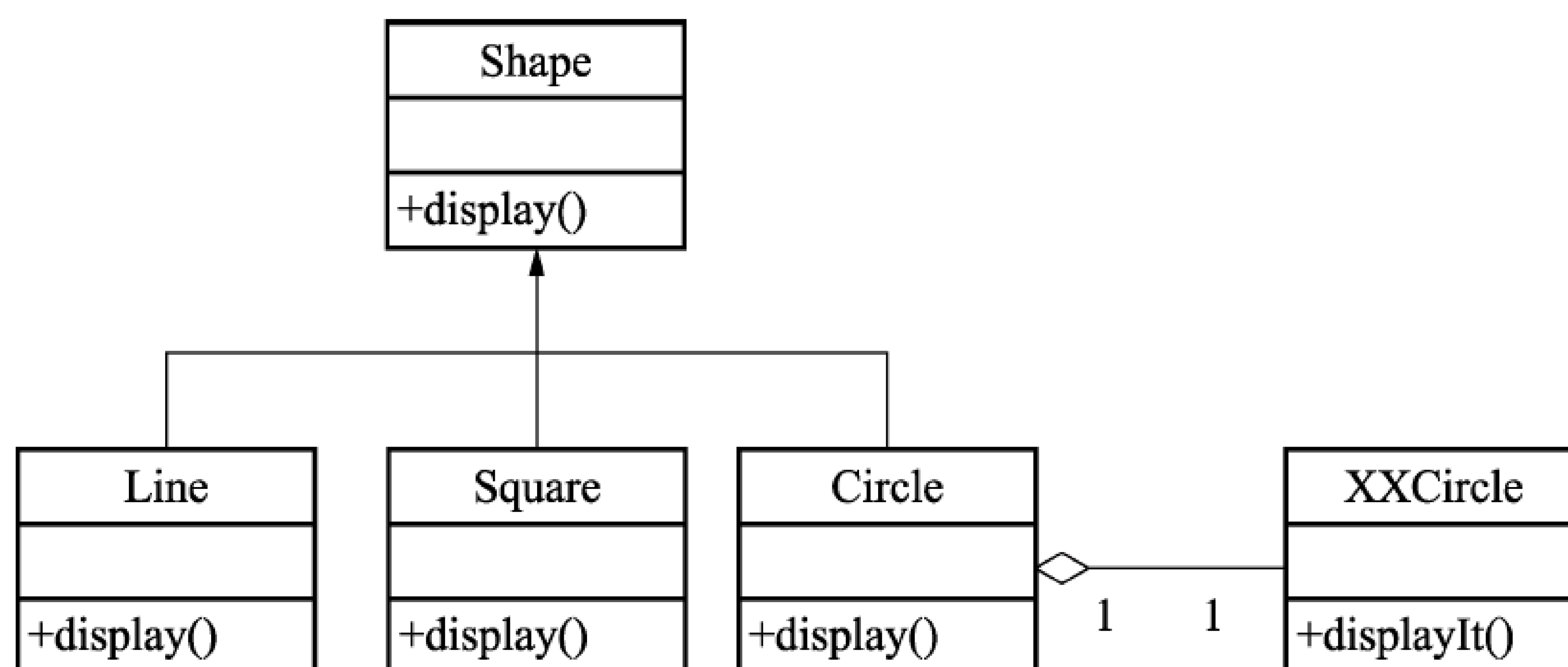


图 7-51　各个类之间的关系

【C++程序】

```
class Shape{
public:
    (1) void display() = 0;
};

class Line:public Shape{//省略具体实现
};

class Square:public Shape{//省略具体实现
};

class XXCircle{
public:
    void displayIt(){
    //省略具体实现
    }
//省略其余方法和属性
};

class Circle:public Shape{
private:
    XXCircle *pxc;
public:
    Circle();
    void display();
};
Circle::Circle(){
    pxc = (2);
}
void Circle::display()
{
    pxc->(3);
}

class Factory{
```

```
public:
    (4) getShapeInstance(int type){ //生成特定类实例
        switch(type){
            case 1 : return new Square;
            case 2 : return new Line;
            case 3 : return new Circle;
            default : return NULL;
        }
    }
};

void main(int argc,char *argv[]){
    if(argc !=2){
        cout<<"error parameters ! "<<endl;
        return;
    }
    int type=atoi(argv[1]);
    Factory factory;
    Shape *s = factory.(5);
    if(s==NULL){
        cout<<"Error get the instance!"<<endl;
        return;
    }
    s->display();
    delete s;
    return;
}
```

试题七(共 15 分)

阅读以下函数说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

现有一个显示系统，要显示的图形有线 Line、矩形 Square，抽象出一个 Shape 类(接口)，有方法显示 display()。

需要新增图形 Circle，又已知有类 XXCircle 实现了所需要实现的功能：显示 displayIt()。为了继承自 Shape 以提供统一接口，又不希望从头开发代码，希望使用 XXCircle。这样将 XXCircle 作为 Circle 的一个属性，即 Circle 的对象包含一个 XXCircle 对象。当一个 Circle 对象被实例化时，它必须实例化一个相应的 XXCircle 对象；Circle 对象收到的做任何事的请求都将转发给这个 XXCircle 对象。通过这种被称为 Adapter 的模式，Circle 对象就可以通过“让 XXCircle 做实际工作”来表现自己的行为了。图 7-52 显示了各个类间的关系。以下是 Java 语言实现，能够正确编译通过。

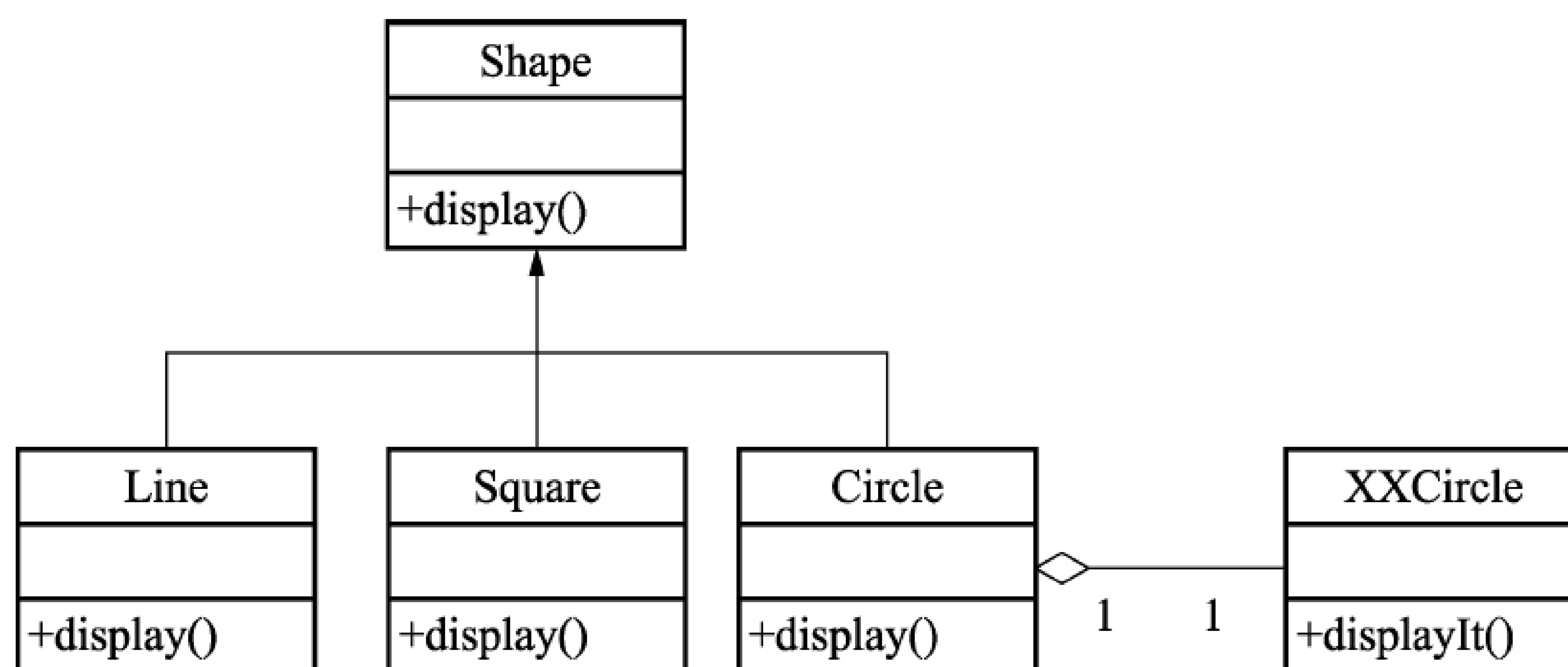


图 7-52　各个类之间的关系

【Java 程序】

```
//Shape.java 文件
public interface Shape {
    public  (1)  void display();
}

//XXCircle.java 文件
public class XXCircle {
    public void displayIt(){
    //省略具体实现
    }
}

//Circle.java 文件
public class Circle  (2)  Shape {
    private XXCircle pcx =  (3) ;
    public void display(){
        pcx.displayIt();
    }
}

//Factory.java 文件
public class Factory {
    public  (4)  getShapeInstance(int type){
        switch(type){
        case 1 : return new Line();
        case 2 : return new Square();
        case 3 : return new Circle();
        default : return null;
        }
    }
}

//Main.java 文件
public class Main {
    public static void main(String[] args) {
        int type=1;
```

```
            Factory factory = new Factory();
            Shape s;
            s = factory.(5);
            if(s==null){
                System.out.println("Error get the instance!");
                return;
            }
            s.display();
            return;
        }
    }
```

7.1.7 样卷七

全国计算机技术与软件专业技术资格(水平)考试样卷七

试题一~试题四是必答题

试题一(共 15 分)

阅读下列说明和数据流图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

考务处理系统具有以下功能。

(1) 对考生送来的报名单进行检查。

(2) 对合格的报名单编好准考证号后将准考证送给考生，并将汇总后的考生名单送给阅卷者。

(3) 对阅卷站送来的成绩清单进行检查，并根据考试中心制定的合格标准审定合格者。

(4) 制作考生通知单送给考生。

(5) 进行成绩分类统计(按地区、年龄、文化程度、职业、考试级别等分类)和试题难度分析，产生统计分析表。

以下是经分析得到的数据流图及部分数据字典，有些地方有待填充，假定顶层数据流图是正确的。图 7-53 是顶层数据流图，图 7-54 是 0 层数据流图，图 7-55 是 1 层数据流图，其中图 7-55(a)是加工 1 的子图，图 7-55(b)是加工 2 的子图。

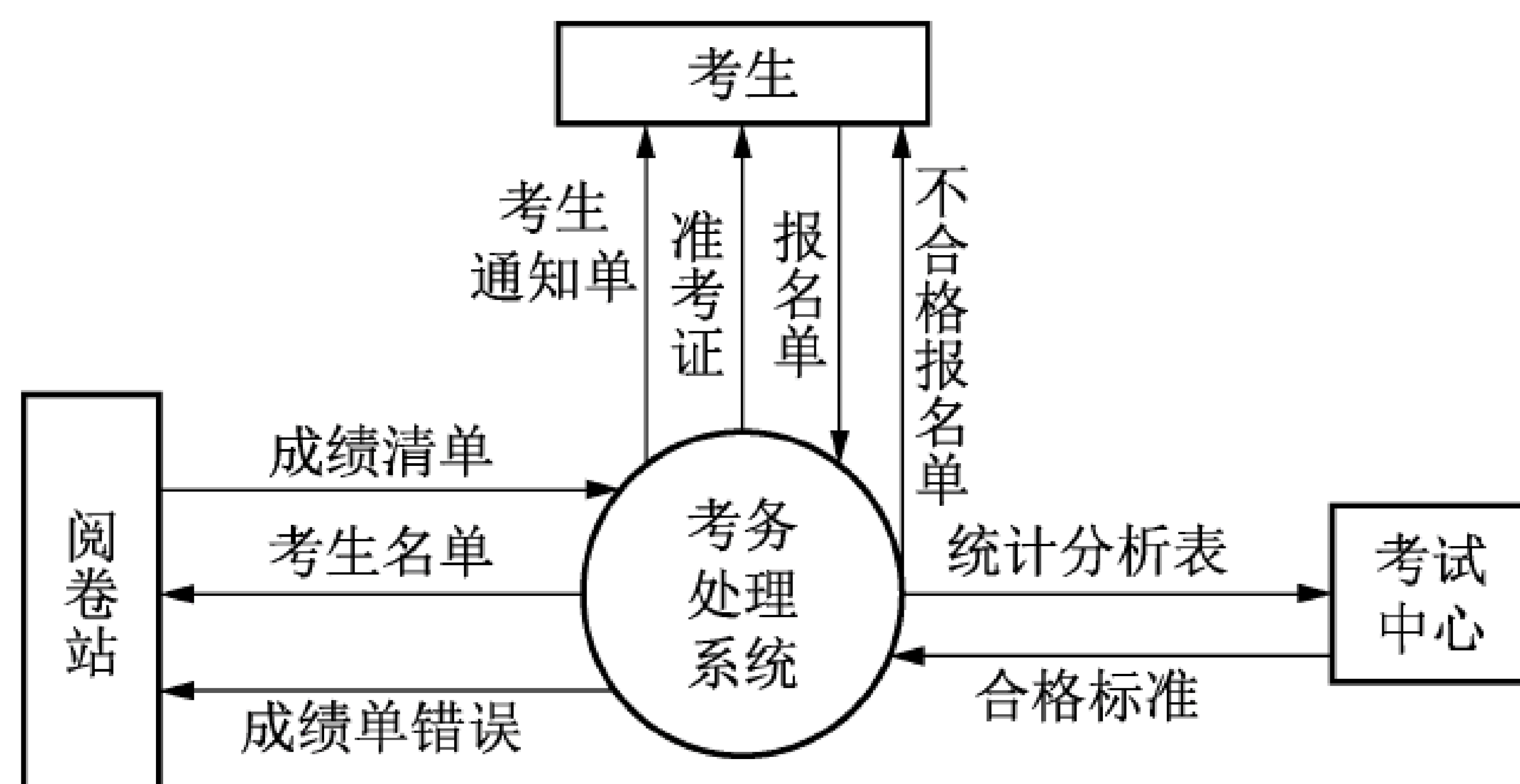


图 7-53 顶层数据流图

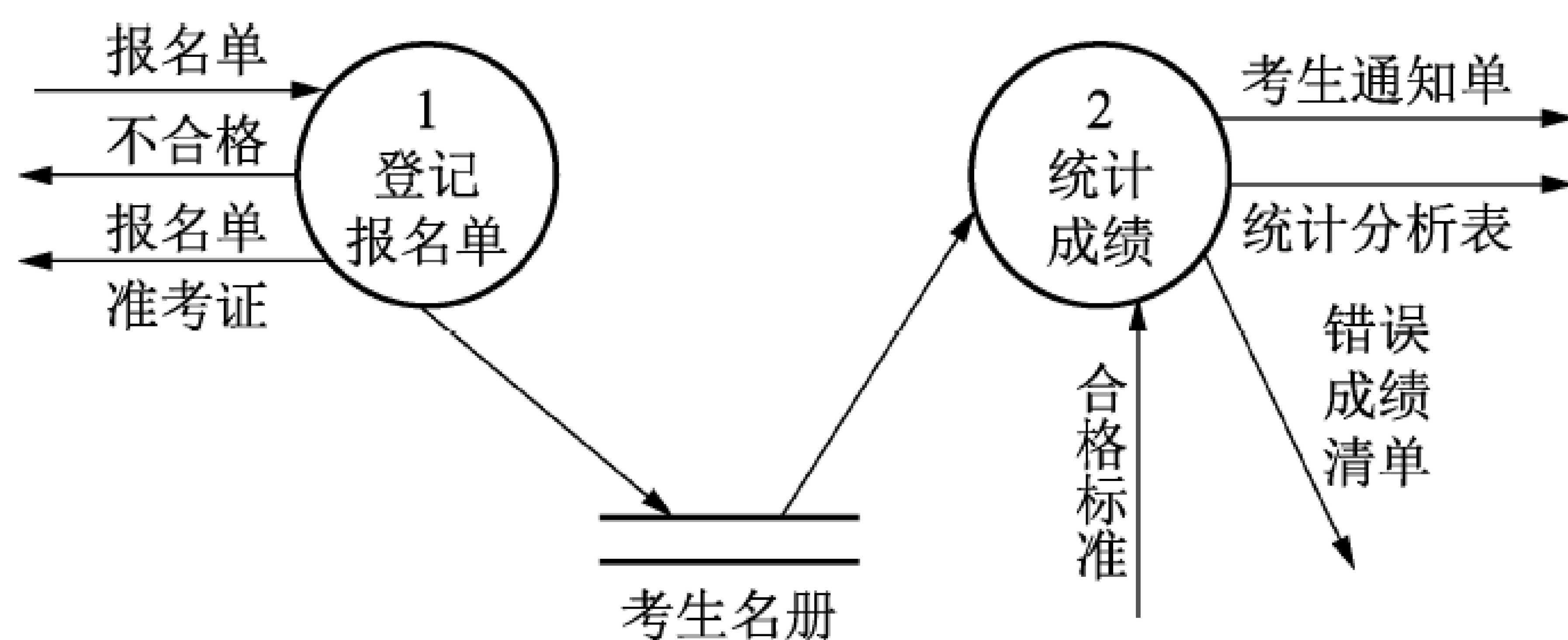
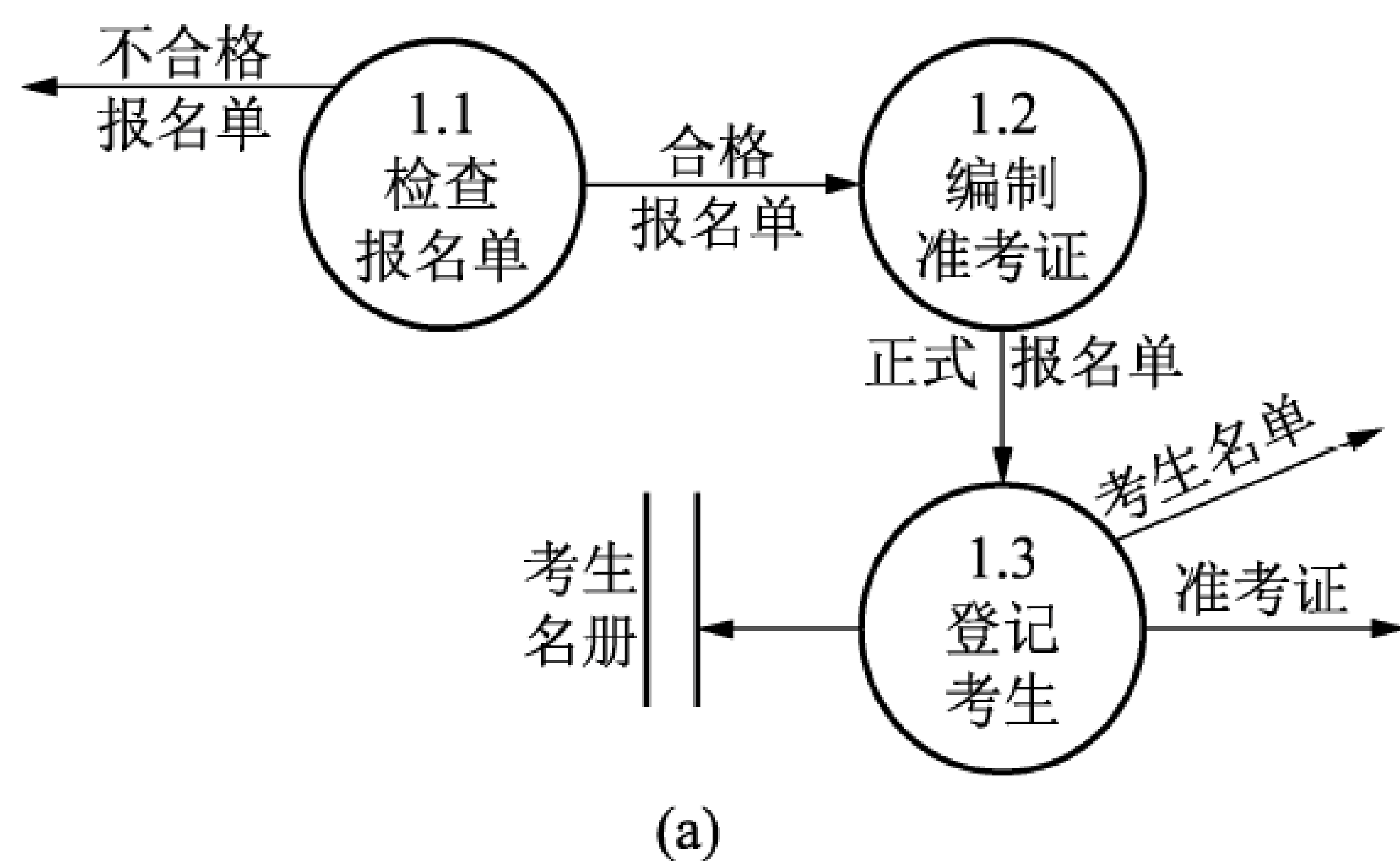
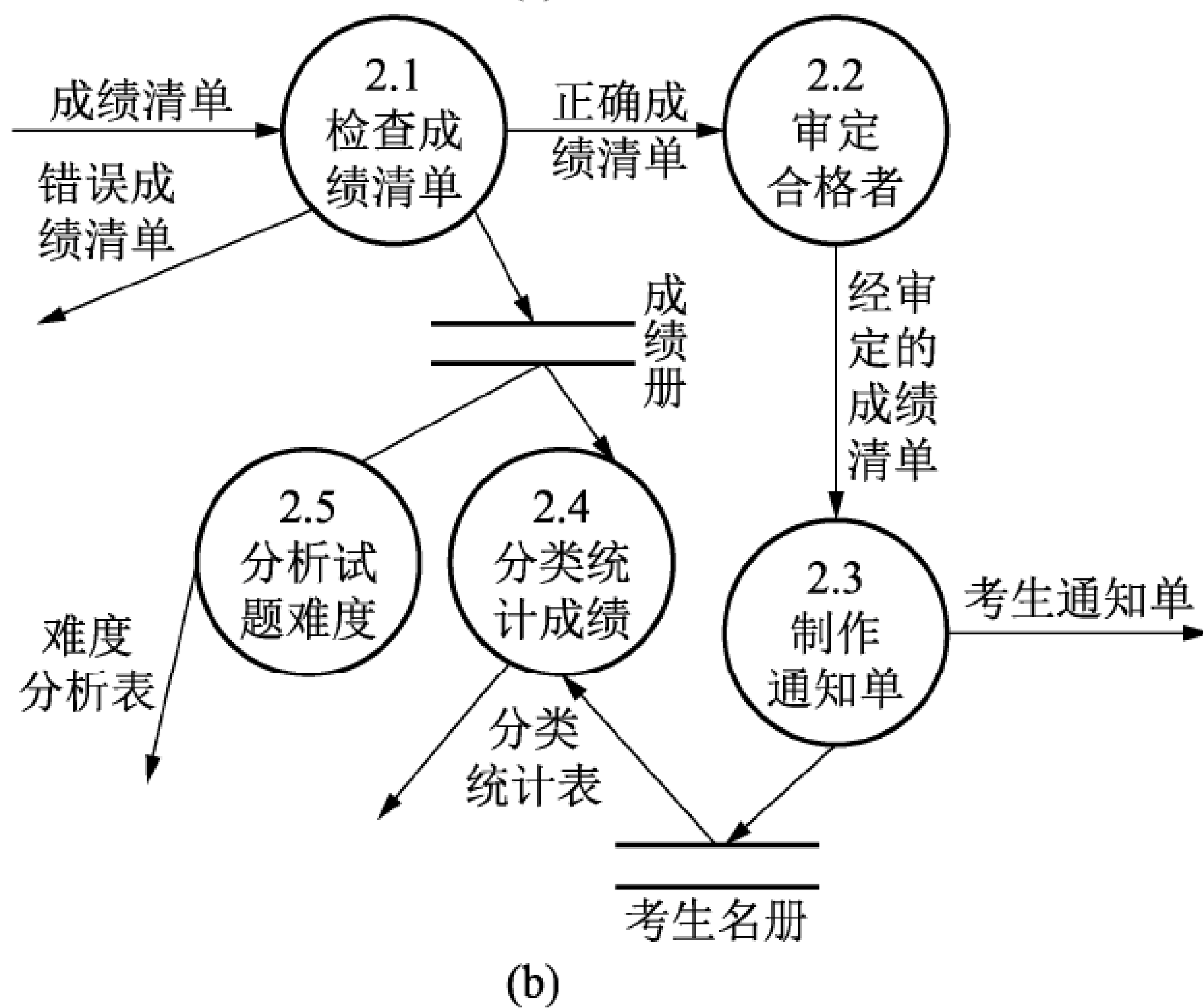


图 7-54　0 层数据流图



(a)



(b)

图 7-55　1 层数据流图

【数据字典】

报名单=地区+序号+姓名+性别+年龄+文化程度+职业+考试级别+通信地址；

正式报名单=报名单+准考证号；

准考证=地区+序号+姓名+准考证号+考试级别；

考生名单={准考证号+考试级别}；

统计分析表=分类统计表+难度分析表；

考生通知单=考试级别+准考证号+姓名+合格标志+通信地址。

【问题 1】(4 分)

根据题意，指出 0 层数据流图(见图 7-54)中缺失的数据流的名称，并指出该数据流的起

点和终点。

【问题 2】(4 分)

根据题意，指出加工 1 子图(见图 7-55(a))中缺失的数据流的名称，并指出该数据流的起点和终点。

【问题 3】(7 分)

根据题意，指出加工 2 子图(图 7-55(b))中缺失的数据流的名称，并指出该数据流的起点和终点。加工 2 子图(图 7-55(b))中有一条数据流是错误的，请指出这条数据流的起点和终点。

试题二(共 15 分)

阅读下列说明和 E-R 图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

图 7-56 是某医院组织的结构图。该医院分为多个病区，每个病区有一个唯一的编号，一个病区包括多个病房、多名医生；每位医生有一个唯一的编号，负责管辖其主治病人的所有病房；病人住院后给予一个唯一的编号，根据“患何种病科”住在相应病区的某个病房里，有且仅有一位医生担任主治医生，除主治医生外其他医生不对其负责。



图 7-56　某医院组织的结构图

现假定病区名称有“内科”和“外科”，“内科”病区又细分为多个病区，以编号区分，名称都为“内科”；“外科”病区亦然。图 7-57 是经分析得到的 E-R 图。

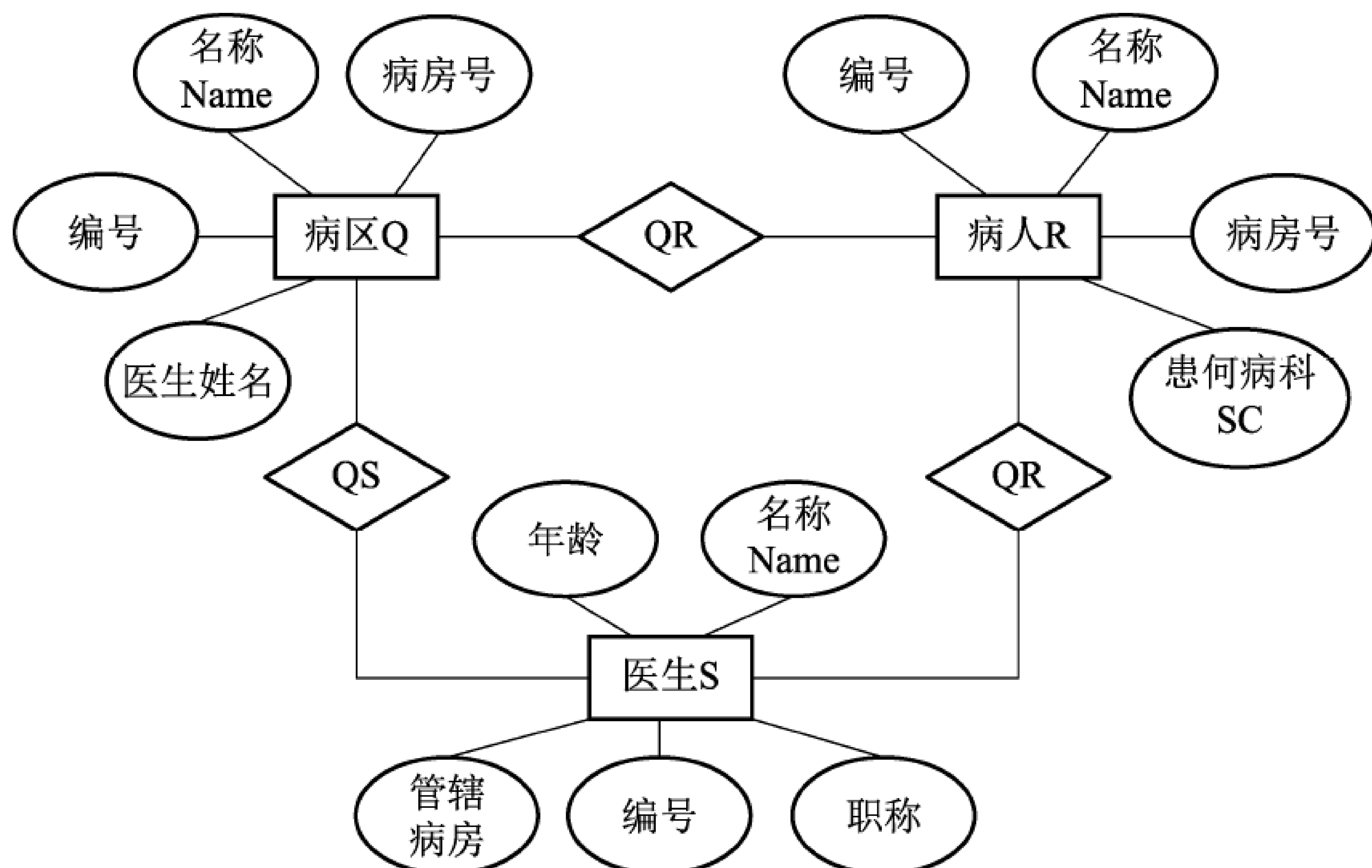


图 7-57　经分析得到的 E-R 图

【问题 1】(6 分)

实体间的联系有“一对一”“一对多”和“多对多”，指出图 7-57 中各联系分别属于哪一种。

【问题 2】(4 分)

选出正确的关系代数表达式。

(1) 查询所有“外科”病区和“内科”病区的所有医生姓名(　　)。

A. $\sigma_{Name="外科"\vee Name="内科"}(\pi_4(Q))$　　B. $\sigma_{Name="外科"\wedge Name="内科"}(\pi_4(Q))$

C. $\pi_4(\sigma_{Name="外科"\vee Name="内科"}(Q))$　　D. $\pi_4(\sigma_{Name="外科"\wedge Name="内科"}(Q))$

(2) 查询内科病区患胃病的病人的姓名(　　)。

A. $\sigma_{Name="内科"\vee SC="胃病"}(\pi_2(R))$　　B. $\sigma_{Name="内科"\wedge SC="胃病"}(\pi_2(R))$

C. $\pi_2(\sigma_{Name="内科"\vee SC="胃病"}(R))$　　D. $\pi_2(\sigma_{Name="内科"\wedge SC="胃病"}(R))$

【问题 3】(6 分)

层次模型不能直接表示多对多联系，为什么？可采用哪些方法进行多对多联系的表示?

试题三(共 15 分)

阅读下列说明和图，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。

【说明】

移动电话是传统固定式电话的延伸，通过无线电网络可以与千里之外的朋友沟通而不受电话线的束缚。现在的移动电话功能更全面，除了作为电话使用外，还可以发送短信，可以管理电话簿，可以下载铃声、图案……

手机由键盘、显示屏及移动通信设备组成，移动通信设备负责发送和接收信号，与基站进行连线。打电话的流程如下。

(1) 用户拨电话号码，每按下一个数字键，显示屏上显示相应数字。

(2) 按 OK 键进行连线，显示屏上显示“连线中……”，请求连接基站，基站通过移动电话网络连接到对方手机，若有误则返回相关信息。

(3) 接通后，显示屏显示“连线成功”。

(4) 打电话结束后，按 Cancel 键送出断线信号，通知移动电话基站断线，基站切断连接，显示屏显示“断线成功”。

该系统采用面向对象方法开发，系统中的类及类之间的关系用 UML 类图表示。图 7-58 是该系统的用例图，图 7-59 是该系统的类图，图 7-60 描述了打电话(包括断开)的序列图。

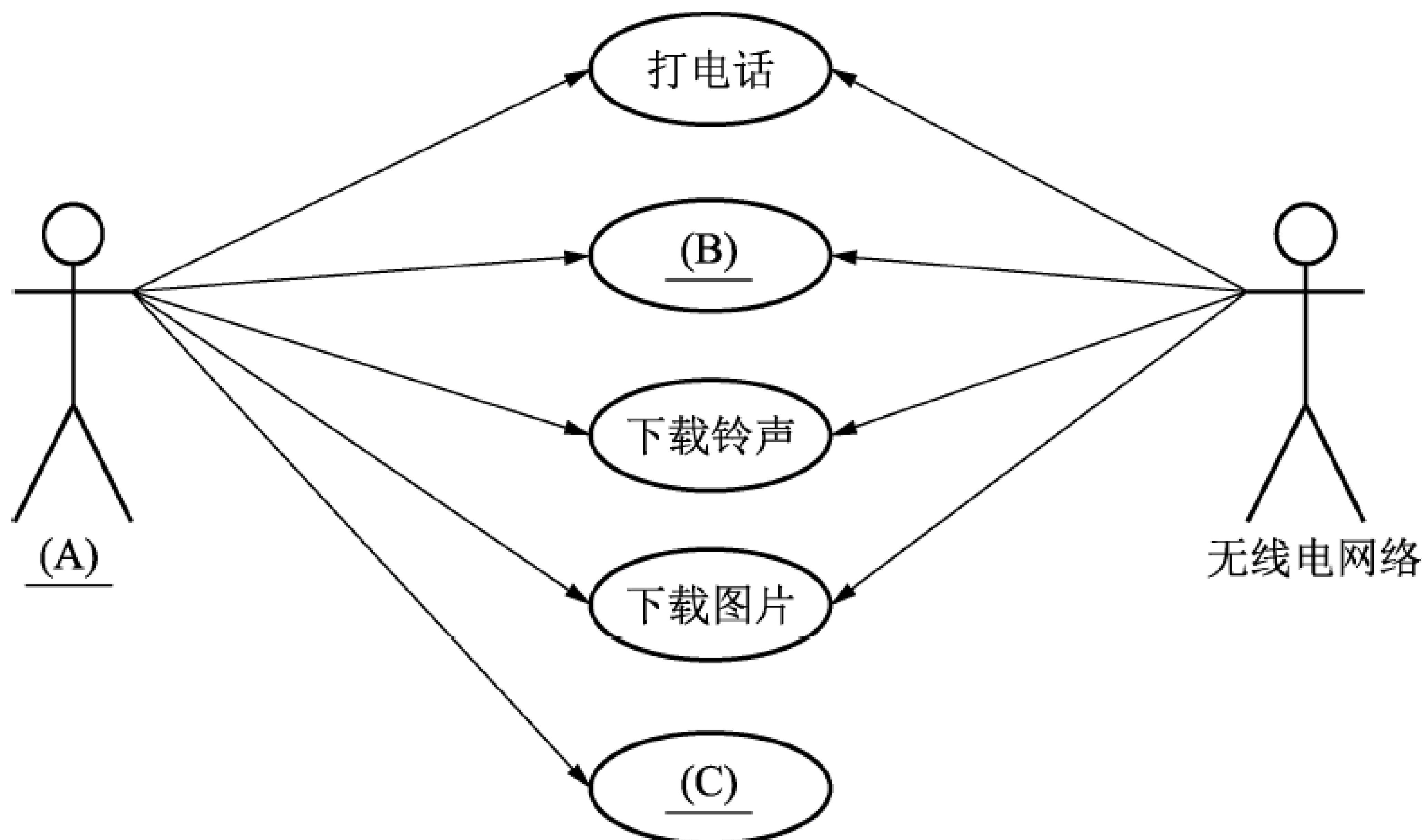


图 7-58　系统用例图

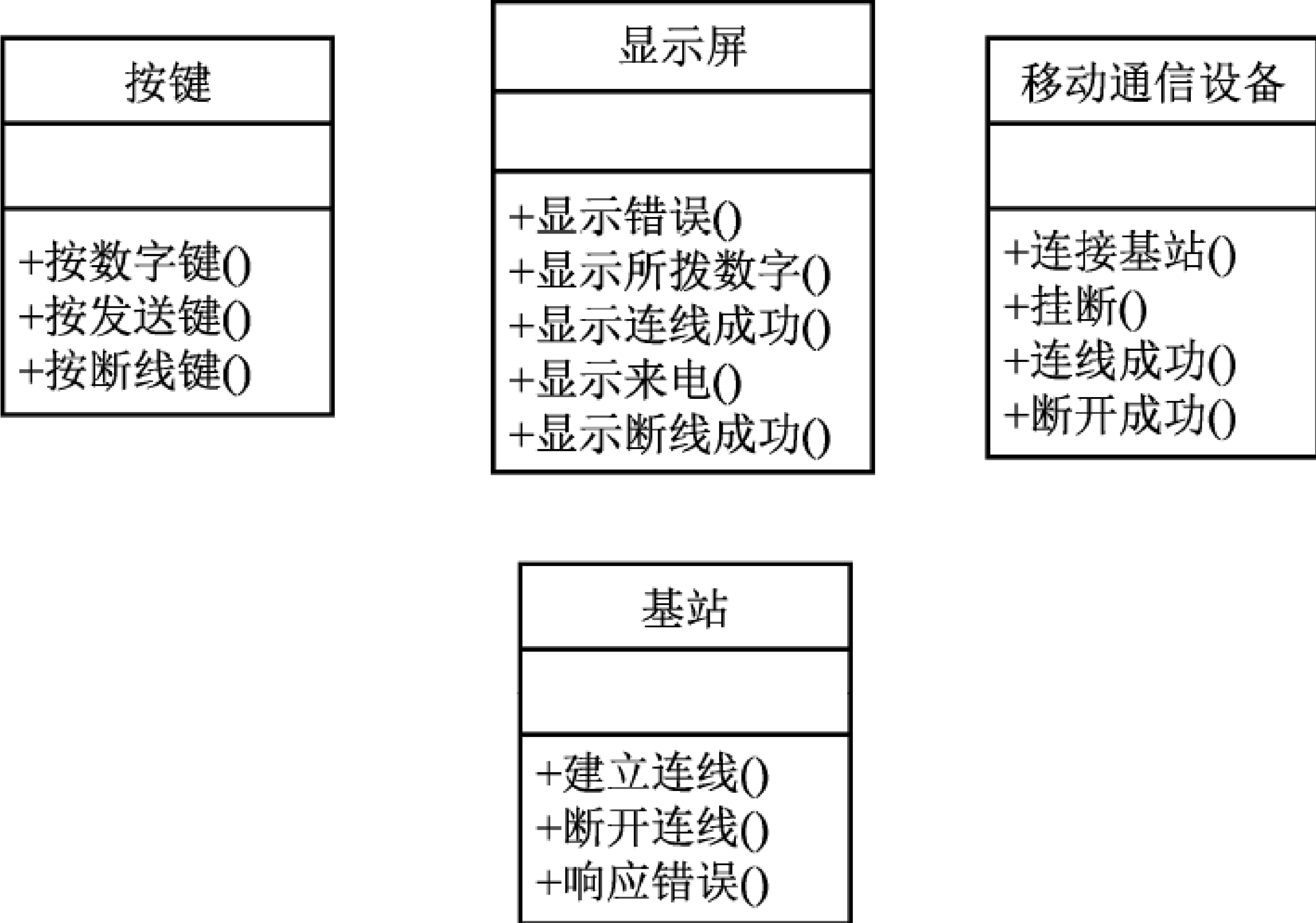


图 7-59 系统类图

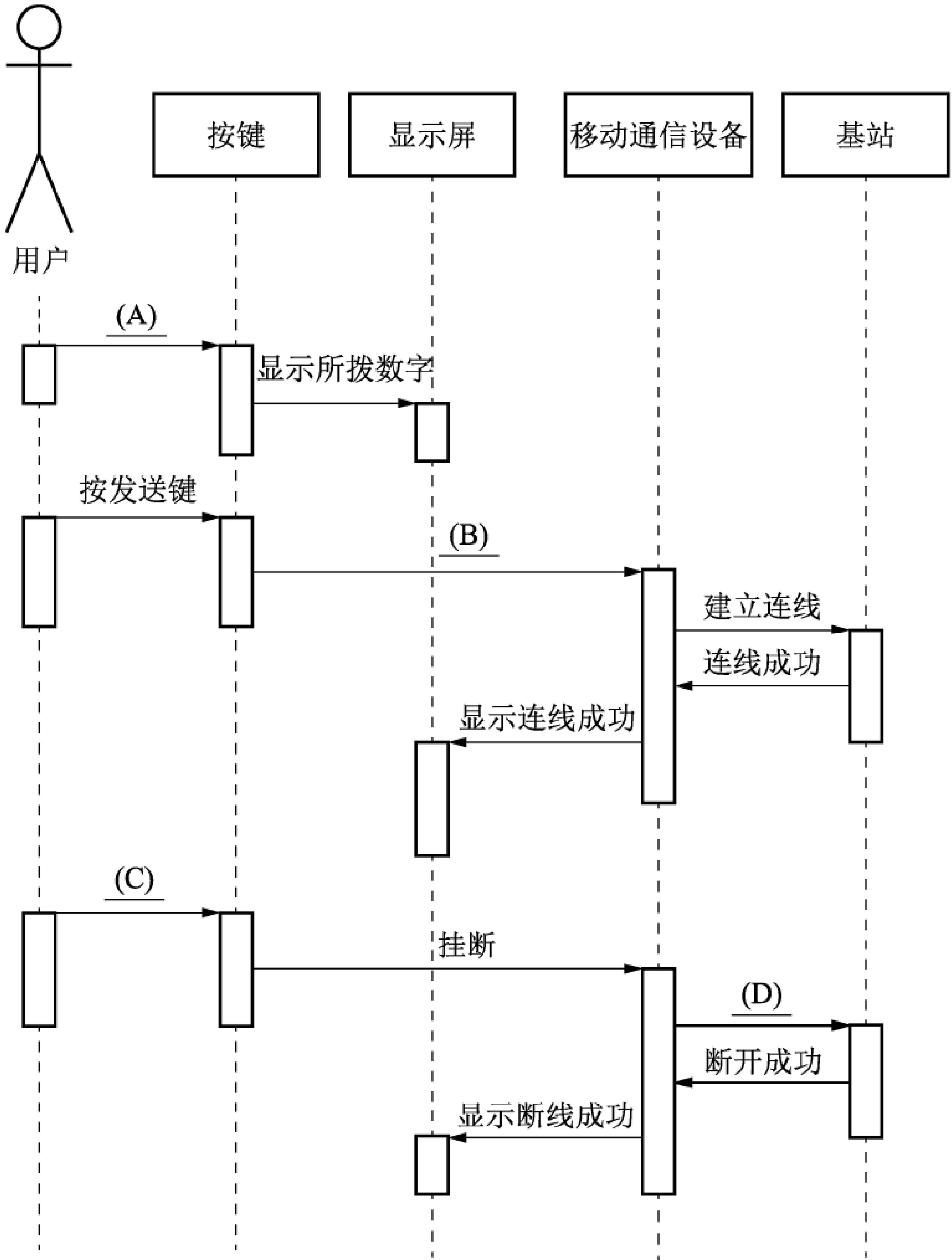


图 7-60 序列图

【问题 1】(7 分)

根据题意，用题中及类图中提供的术语指出图 7-58 中的参与者(A)及用例(B)、(C)各是什么。

【问题 2】(8 分)

根据题意，用题中及类图中提供的术语指出图 7-60 所示的打电话序列图中的消息。

试题四(共 15 分)

阅读下列说明和图表，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

在多道程序系统中，各个程序之间是并发执行的，共享系统资源。CPU 需要在各个运行的程序之间来回地切换，这样的话，要想描述这些多道的并发活动过程就变得很困难。为此，操作系统设计者提出了进程及进程的概念。

进程是具有独立功能的程序关于某个数据集合上的一次动态执行过程，是系统进行资源分配和调度的独立单位。

【问题 1】(3 分)

进程在生命消亡前处于且仅处于 3 种基本状态之一。运行态(Running)：进程占有 CPU，并在 CPU 上运行。就绪态(Ready)：一个进程已经具备运行条件，但由于无 CPU 暂时不能运行的状态(当调度给其 CPU 时，立即可以运行)。等待态(Blocked)：进程因等待某种事件的发生而暂时不能运行的状态，即使 CPU 空闲，该进程也不可运行。

指出如图 7-61 所示的进程状态转换图中“状态 1”~“状态 3”分别是什么状态。

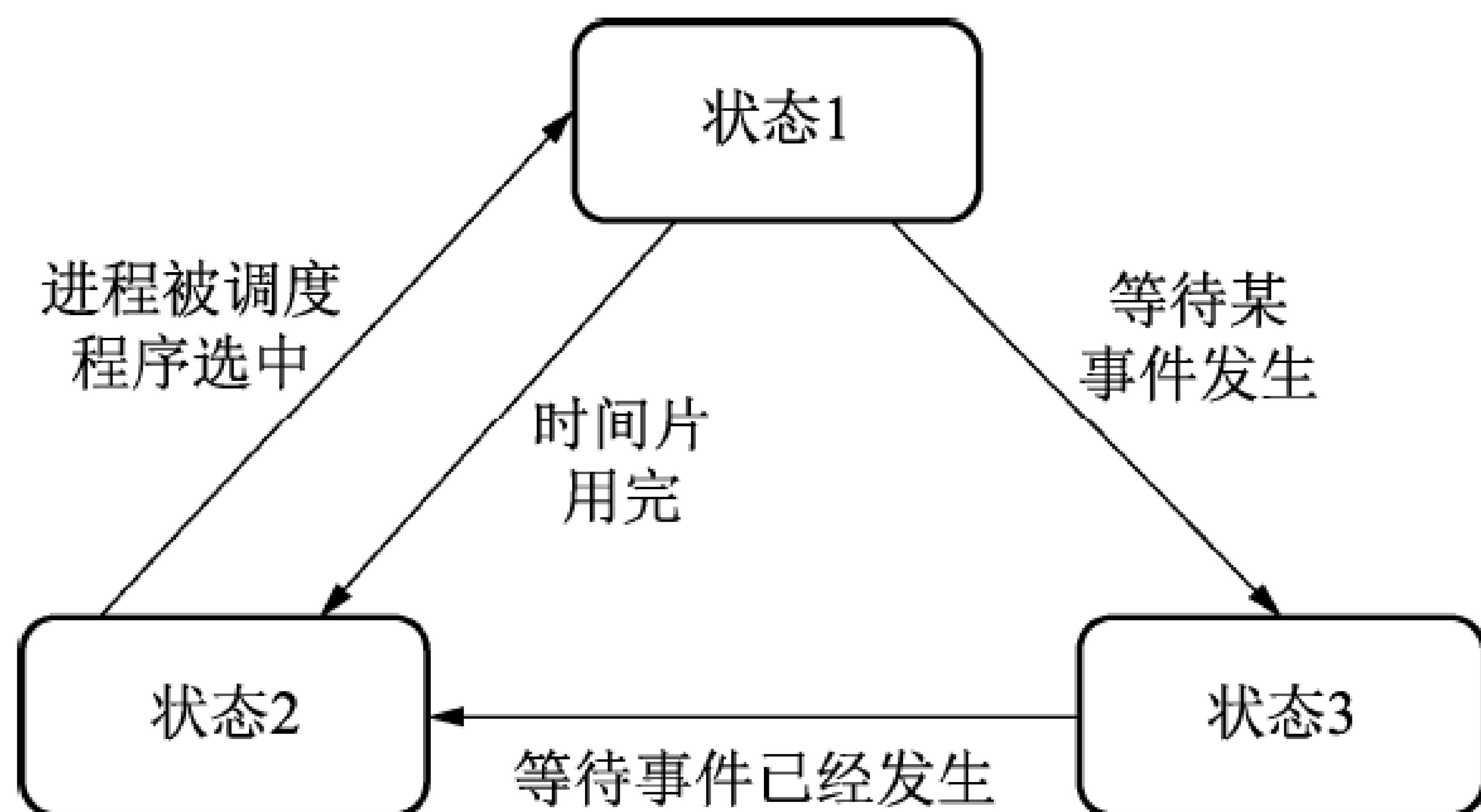


图 7-61　状态转换图

【问题 2】(6 分)

如果单 CPU 系统中有 N 个进程，运行的进程最多有几个，最少有几个？就绪进程最多有几个，最少有几个？等待进程最多有几个，最少有几个？

【问题 3】(6 分)

进程调度算法解决以何种次序对各就绪进程进行处理机的分配以及按何种时间比例让进程占用处理机的问题。

常见的调度算法有：先进先出 FIFO(按照进程进入就绪队列的先后次序选择)、时间片轮转 RR(进程轮流运行一个时间片)、最高优先级 HPF(分配给具有最高优先级的就绪进程)。

在实际系统中，调度模式往往是几种调度算法的结合。某系统按优先级别设置若干个

就绪队列，对级别较高的队列分配较小的时间片 $S_i(i=1,2,\cdots,n)$，即有 $S_1<S_2<\cdots<S_n$。除第 n 级队列是按 RR 法调度之外，其他各级队列均按 FIFO 调度。系统总是先调度级别较高的队列中的进程，仅当该队列为空时才去调度下一级队列中的进程。当执行进程用完其时间片时，它便被剥夺并进入下一级就绪队列。当等待进程被唤醒时，它进入其优先级相应的就绪队列，若其优先级高于执行进程，便抢占 CPU 执行进程。

现有 5 个进程 P1、P2、P3、P4、P5，它们同时依次进入就绪队列，它们所需的 CPU 时间和优先级如表 7-8 所示。注意，优先数越大优先级越低。

表 7-8　CPU 的时间和优先级

进　程	CPU 时间	优 先 级
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

在该系统中，假定不同级别的时间片为 $S_i=2^{i-1}$ (i 为优先数)，请给出 5 个进程的 CPU 占用序列，并注明每次占用所用的时间。

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(共 15 分)

阅读下列函数说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【函数说明】

所谓货郎担问题，是指给定一个无向图，并已知各边的权，在这样的图中，要找一个闭合回路，使回路经过图中的每一个点，而且回路各边的权之和最小。

应用贪婪法求解该问题，程序先计算由各点构成的所有边的长度(作为边的权值)，按长度大小对各边进行排序后，按贪婪准则从排序后的各边中选择边组成回路的边，贪婪准则使得边的选择按各边长度从小到大选择。

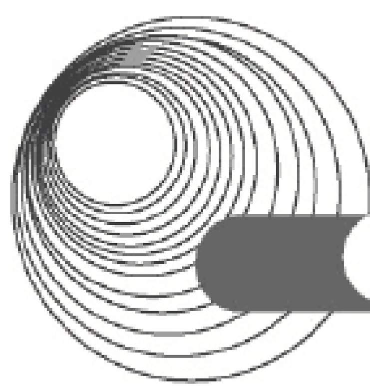
函数中使用的预定义符号如下。

```
#define M 100
typedef struct{/*x 为两端点 p1、p2 之间的距离，即 p1、p2 所组成边的长度*/
    float x;
    int p1, p2;
}tdr;
typedef struct{/*p1、p2 为和端点相联系的两个端点，n 为端点的度*/
    int n, p1, p2;
}tr;
typedef struct{/*给出两点坐标*/
    float x, y;
```

```
}tpd;
typedef int tl[M];
int n = 10;
```

【C 程序】

```
    float distance(tpd a, tpd b);/*计算端点 a、b 之间的距离*/
    void sortArr(tdr a[M], int m);
    /*将已经计算好的距离关系表按距离大小从小到大排序形成排序表，m 为边的条数*/
    int isCircuit(tr r[M], int i, int j);
    /*判断边(i,j)选入端点关系表 r[M]后，是否形成回路，若形成回路返回 0*/
    void selected(tr r[M], int i, int j);/*边(i,j)选入端点关系表 r*/
    void course(tr r[M], tl l[M]);/*从端点关系表 r 中得出回路轨迹表*/
    void exchange(tdr a[M], int m, int b);
    /*调整排序表，b 表示是否可调，即是否有边长度相同的边存在*/
    void travling(tpd pd[M], int n, float dist, tl locus[M])
    /*dist 记录总路程*/
    {
        tdr dr[M];/*距离关系表*/
        tr r[M];/*端点关系表*/
        int i, j, k, h, m;/*h 表示选入端点关系表中的边数*/
        int b;/*标识是否有长度相等的边*/
        k = 0;
        /*计算距离关系表中各边的长度*/
        for(i = 1; i < n; i++){
            for(j = i + 1; j <= n; j++){
                k++;
                dr[k].x = __(1)__;
                dr[k].p1 = i;
                dr[k].p2 = j;
            }
        }
        m = k;
        sortArr(dr, m);/*按距离大小从小到大排序形成排序表*/
        do{
            b = 1;
            dist = 0;
            k = h = 0;
            do{
                k++;
                i = dr[k].p1;
                j = dr[k].p2;
                if((r[i].n <= 1) && (r[j].n <= 1)){/*度数不能大于 2*/
                    if(__(2)__){
                        /*若边(i,j)加入 r 后形成回路，则不能加入*/
                        __(3)__;
                        h++;
                        dist += dr[k].x;
                    }else if(__(4)__){
                        /*最后一边选入 r 构成回路，则该边必须加入且得到解*/
                        selected(r, i, j);
```




```
                    h++;
                    dist += dr[k].x;
                }
            }
        }while((k != n) && (h != n));
        if(h == n){/*最后一边选入 r 构成回路，完成输出结果*/
            course(r, locus);
        }else{/*找不到解，调整 dr，交换表中边长相同的边在表中的顺序，并将 b 置 0*/
            (5);
        }
    }while(!b);
}
```

7.1.8 样卷八

全国计算机技术与软件专业技术资格(水平)考试样卷八

试题一~试题四是必答题

试题一(共 15 分)

阅读下列说明和数据流图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

某供销系统接受顾客的订货单，当库存中某配件的数量小于订购量或库存量低于一定数量时，向供应商发出采货单；当某配件的库存量大于或等于订购量时，或者收到供应商的送货单并更新了库存后，向顾客发出提货单。该系统还可随时向总经理提供销售和库存情况表。

以下是经分析得到的数据流图及部分数据字典，有些地方有待填充，假定顶层数据流图是正确的。图 7-62 是顶层数据流图，图 7-63 是 0 层数据流图，图 7-64 是 1 层数据流图，其中图 7-64(a)是加工 1 的子图，图 7-64(b)是加工 2 的子图。

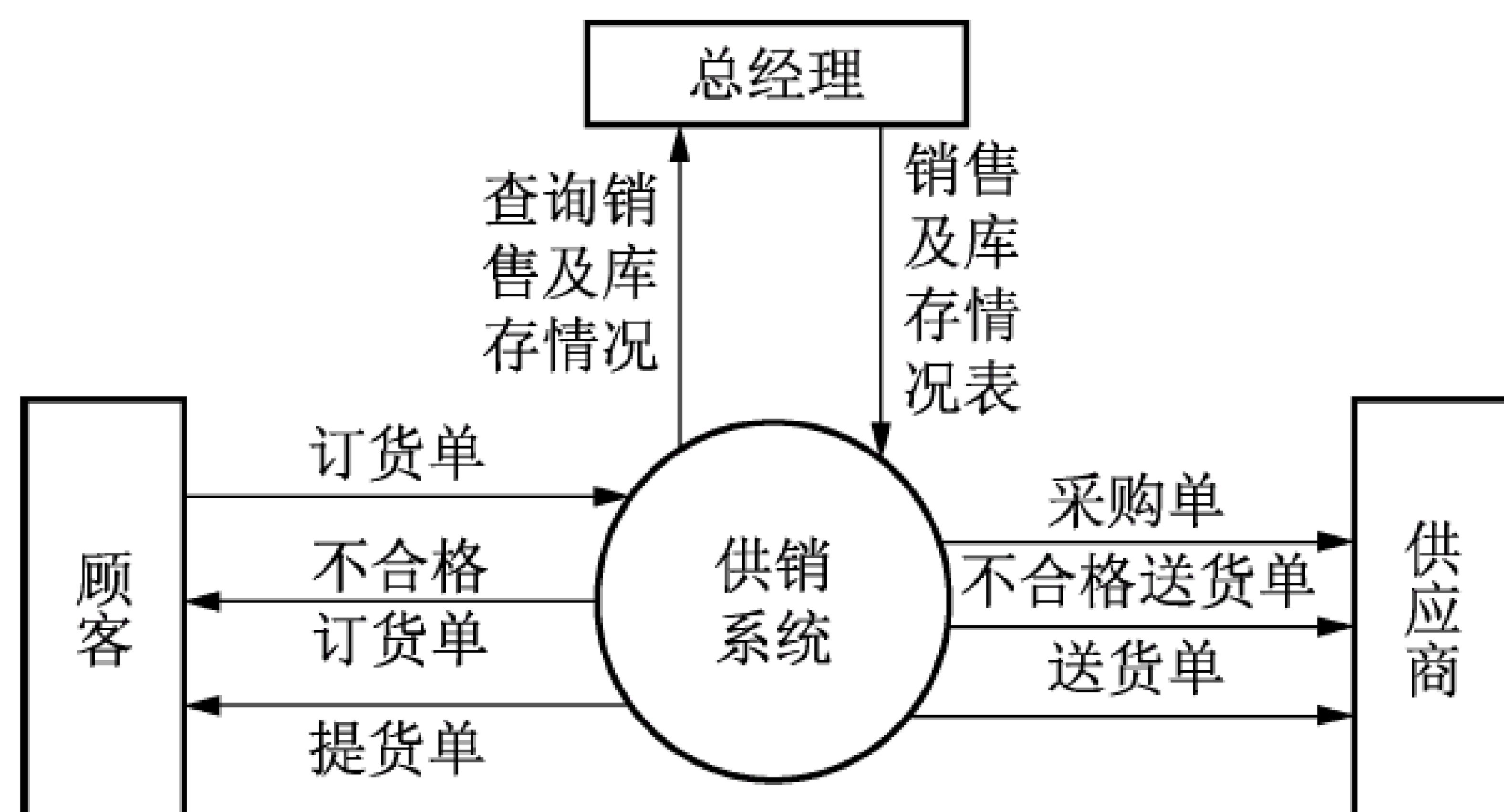


图 7-62　顶层数据流图

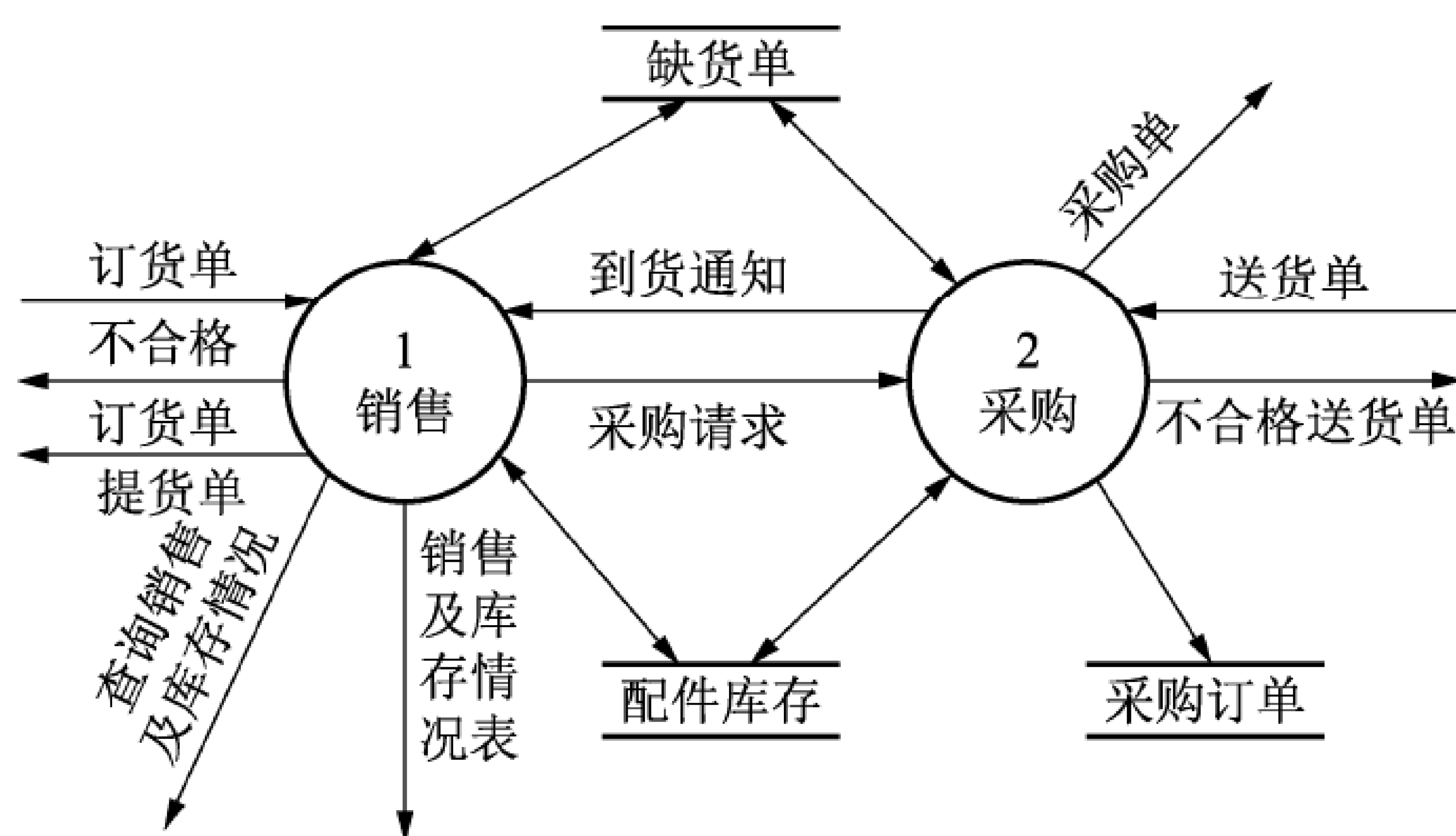
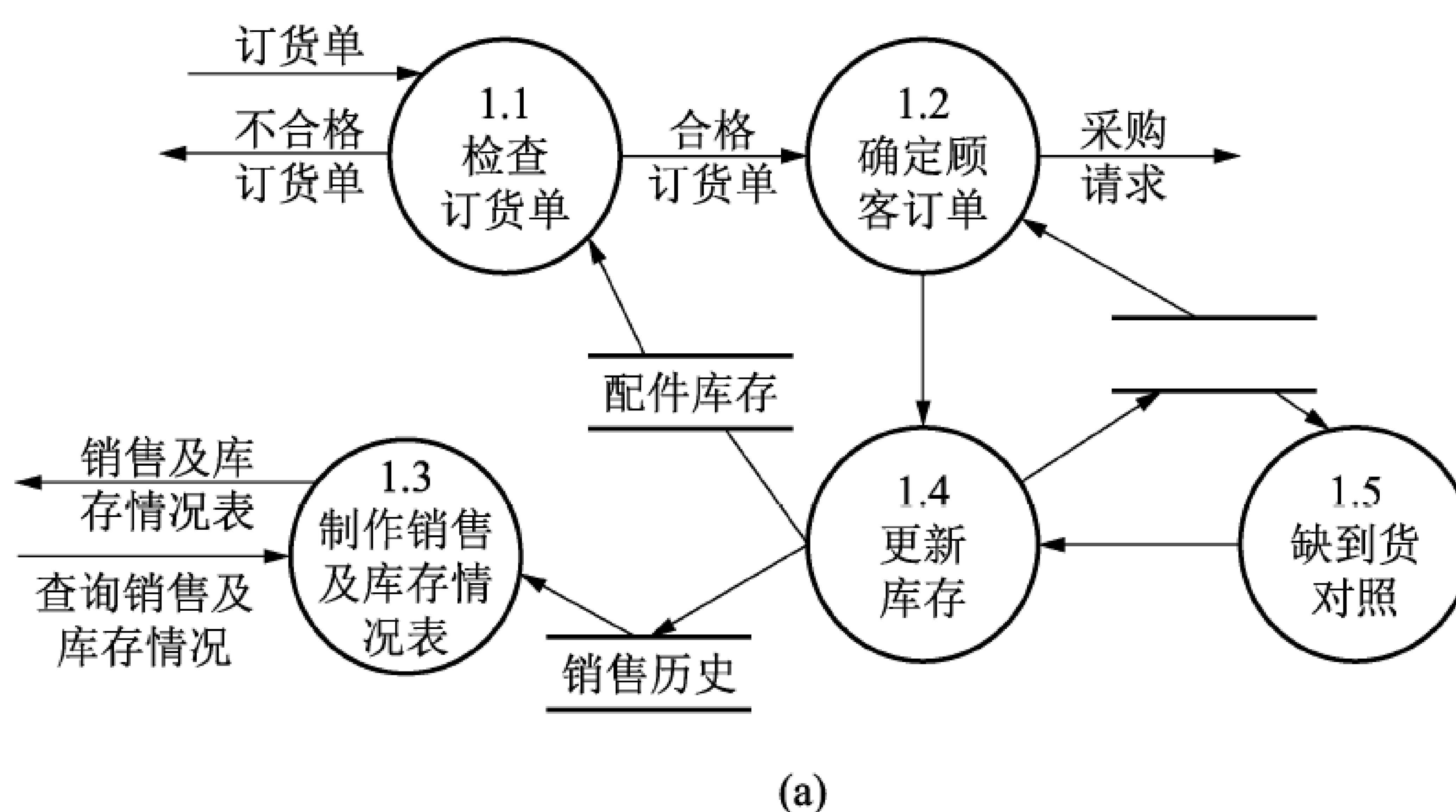
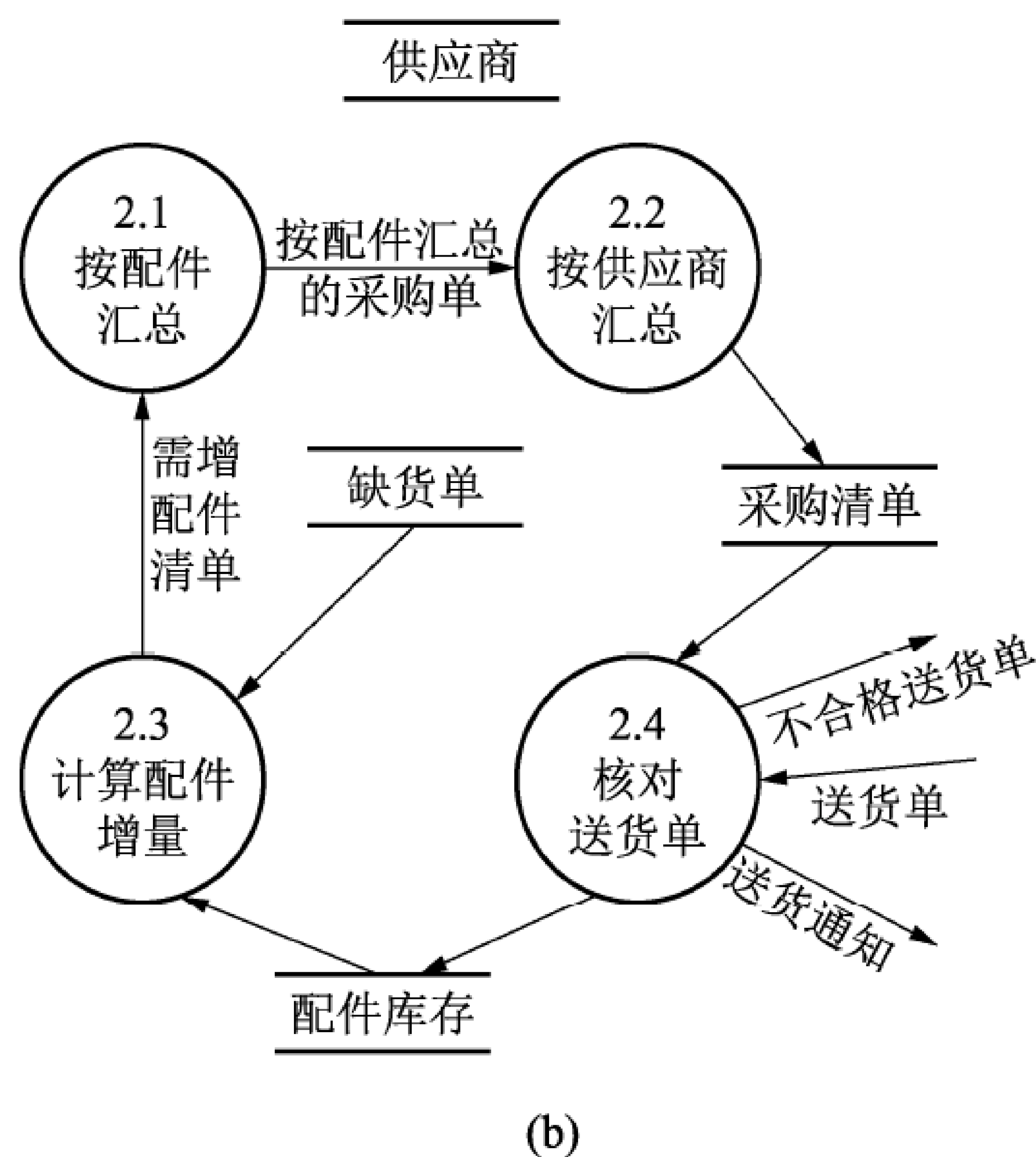


图 7-63　0 层数据流图

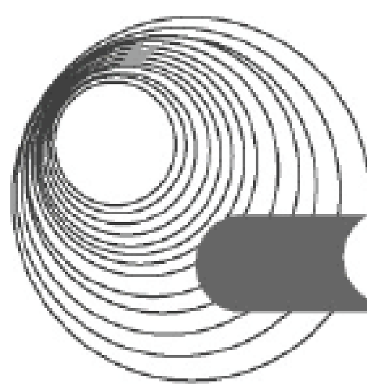


(a)



(b)

图 7-64　第 1 层数据流图



【数据字典】

1) 数据流条目

订货单=配件号+配件名+规格+数量+顾客名+地址；

提货单=订货单+金额；

采货单=配件号+配件名+规格+数量+供应商名+地址；

送货单=配件号+配件名+规格+数量+金额。

2) 文件说明

文件名：配件库存。

组成：{配件号+配件名+规格+数量+允许的最低库存量}。

【问题 1】(3 分)

根据题意，图 7-63 中哪个文件可不必画出？

【问题 2】(8 分)

根据题意，指出图 7-64(a)中缺失的数据流的名称，并指出该数据流的起点和终点。

【问题 3】(4 分)

根据题意，指出图 7-64(b)中缺失的数据流的名称，并指出该数据流的起点和终点。

试题二(共 15 分)

阅读下列说明和 E-R 图，回答问题 1~问题 3，将解答填入答题纸的对应栏内。

【说明】

某学校的教学系统描述如下。

学生信息包括学号(SNo)、姓名(Sname)、性别(Sex)、年龄(Age)、入学年份(Year)、主修专业(Major)，其中学号是入学时唯一编定的。

课程信息包括课程号(CNo)、课程名称(CName)、学时(Period)、学分(Credit)，其中课程号是唯一编定的。

一个学生可选多门课，每个学生选每门课有一个成绩。图 7-65 是经分析得到的 E-R 图。

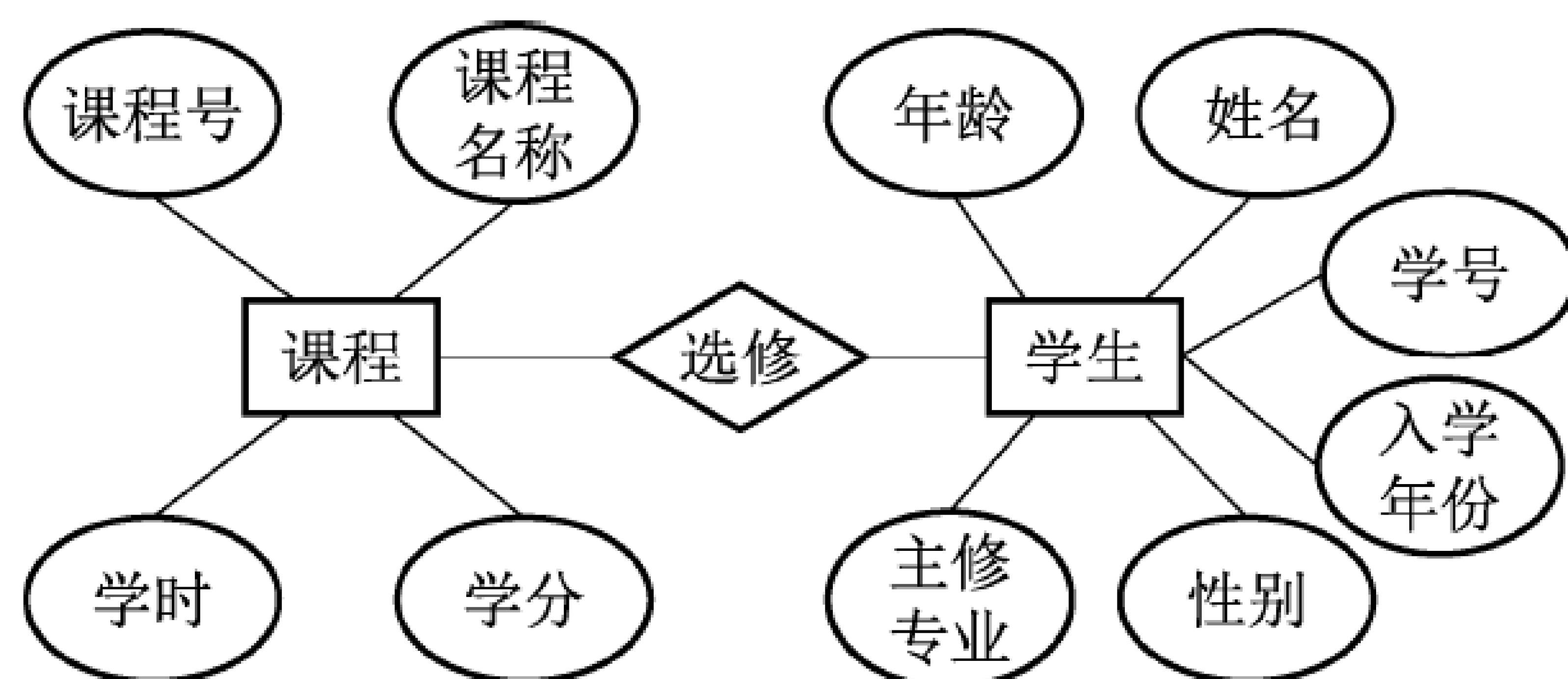


图 7-65　E-R 图

【问题 1】(5 分)

设基本表有 Student(SNo, SName, Sex, Age, Year, Major)、Course(CNo, Cname, Period, Credit)、Grade(SNo, CNo, Grade)，通过以下 SQL 语句建立，请在 SQL 语句空缺处填入正确的内容。

```
CREATE TABLE Student(SNo CHAR(6) NOT NULL,
                     SName CHAR(20),
```

```
                        Sex CHAR(1),
                        Age INTEGER,
                        Year CHAR(4),
                        Major CHAR(20),
                         (1) );
CREATE TABLE Course(CNo CHAR(6) NOT NULL,
                        CName CHAR(20),
                        Period INTEGER,
                        Credit INTEGER,
                         (2) );
CREATE TABLE Grade(SNo CHAR(6) NOT NULL,
                        CNo CHAR(6) NOT NULL,
                        Grade REAL,
                         (3) ,
                         (4) ,
                         (5) );
```

【问题 2】(6 分)

若另有表 Teach(CName, TName)存储教师任课情况，TName 表示教师名。用 SQL 创建一个含有学号、姓名、课程名、成绩、任课教师名的“主修专业为计算机 CS”的学生成绩视图，并要求进行修改、插入操作时保证该视图只有计算机系的学生。请在 SQL 语句空缺处填入正确的内容。

```
CREATE VIEW SG  (6) 
             SELECT Student.SNo, SName, Grade, Course.CName, TName
             FROM Student, Grade, Teach,
             WHERE  (7) 
             AND  (8) 
             AND Major = 'CS',
              (9) ;
```

【问题 3】(4 分)

以下的 SQL 语句是用于查询“每个学生的选修课程数、总成绩、平均成绩”的不完整语句，请在空缺处填入正确的内容。

```
SELECT Student.SNo,  (10) , SUM(Grade), AVG(Grade)
             FROM Student, Grade
             WHERE Student.SNo = Grade.SNo,
             GROUP BY  (11) ;
```

试题三(共 15 分)

阅读下列说明和图，回答问题 1 和问题 2，将解答填入答题纸的对应栏内。

【说明】

银行的自动柜员机(ATM)的功能描述如下。

(1) 金融卡与信用卡识别：包含伪卡识别及密码验证。

(2) 主菜单项：这是一台 ATM 最主要的人机界面，提供各项功能给客户，具体有提款、转账、更改密码及存款。

(3) 结束操作：客户执行完“菜单项”的功能后，可以选择“打印单据”或“不打印单据”，选好后就结束此次交易。

注意：ATM 除了能处理本行的银行卡外，其他银行的银行卡也应该能处理，可以通过“金融中心”与其他银行主机进行数据交换。另外，为了方便，ATM 还提供快捷提款，并提供代缴费功能(代缴费是通过转账方式处理的)。

该系统采用面向对象方法开发，系统中的类及类之间的关系用 UML 类图表示。

【问题 1】(8 分)

图 7-66 是该系统的用例图，根据题意，用题中所述术语指出图 7-66 中参与者(A)、(B)分别是什么？用例(C)、(D)分别是什么？

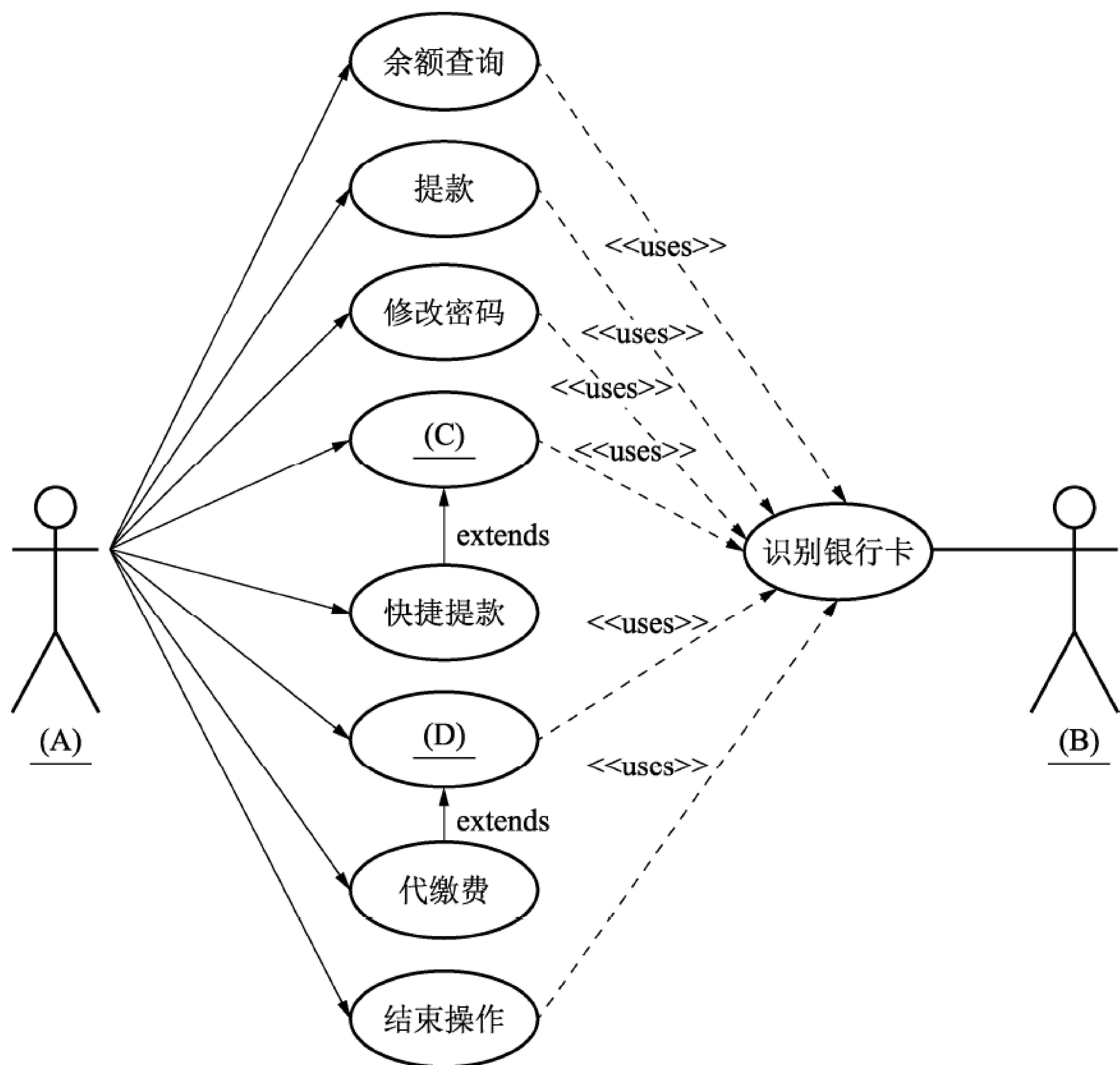


图 7-66　系统用例图

【问题 2】(7 分)

ATM 有以下状态：空闲、银行卡验证、业务选择等待、取款金额输入、密码修改、出钞、单据打印。ATM 一般处于空闲状态，当有客户插入银行卡，则进行银行卡验证，若银行卡无效则结束服务，否则进入业务选择等待状态。业务有取款、修改密码等，也可以选择退出结束服务，ATM 返回空闲状态。选择取款业务后，等待取款金额输入，确认后判断余额是否足够，若余额不足，则给出提示信息，并进入业务选择等待状态；若余额充足，则出钞。若客户需要打印单据，则进入单据打印状态，否则返回业务选择等待状态。选择任意一个业务后，可以取消并返回业务选择等待状态。图 7-67 描述了 ATM 状态的转变情况。

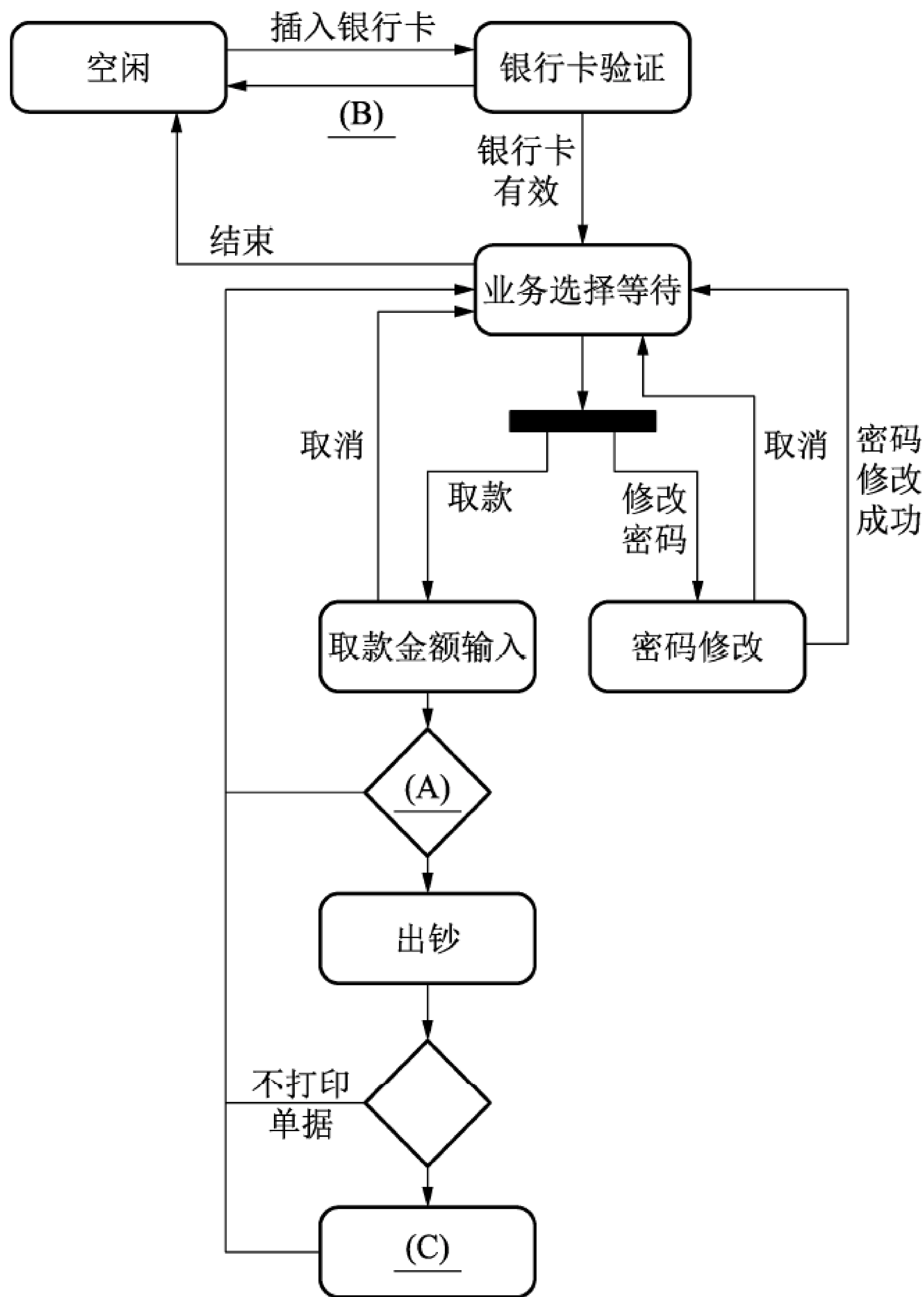


图 7-67　ATM 状态的转变情况

请指出判定(A)、转换(B)及状态(C)分别是什么。

试题四(共 15 分)

阅读下列程序说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【程序说明】

设某城市有 n 个车站，并有 m 条公交线路连接这些车站，设这些公交车都是单向的，这 n 个车站被顺序编号为 0~n−1。本程序输入该城市的公交线路数、车站个数，以及各公交线路上的各站编号，求得从站 0 出发乘公交车至站 n−1 的最少换车次数。

程序利用输入信息构建一张有向图 G(用邻接矩阵 **g** 表示)，有向图的顶点是车站，若有某条公交线路经 i 站能到达 j 站，就在顶点 i 到顶点 j 之间设置一条权为 1 的有向边<i, j>。这样，从站点 x 至站点 y 的最少上车次数便对应图 G 中从点 x 至点 y 的最短路径长度。而程序要求的换车次数就是上车次数减 1。

【C 程序】

```
#include <stdio.h>
#define  M  20
```

```
#define  N  50
int a[N+1];   /*用于存放一条线路上的各站编号*/
int g[N][N];  /*存储对应的邻接矩阵*/
int dist[N];  /*存储站 0 到各站的最短路径*/
int m, n;
void buildG( )
{
    int i, j, k, sc, dd;
    printf("输入公交线路数，公交站数\n");
    scanf("%d%d", &m, &n);
    for( i = 0; i < n; i++)  /* 邻接矩阵清 0 */
        for(j = 0; j < n; j++)g[i][j] = 0;
    for( i = 0; i < m; i++) {
        printf("沿第 %d 条公交车线路前进方向的各站编号(0<=编号<=%d,-1 结束):\n",
            i+1, n-1);
        sc = 0; /* 当前线路站计数器 */
        while(1){
            scanf ("%d", &dd);
            if (dd == -1) break;
            if (dd >= 0 && dd < n) (1) ;
        }
        a[sc] = -1;
        for(k = 1; a[k] >= 0; k++)  /* 处理第 i+1 条公交线路 */
            for (j = 0; j < k; j++)
                g (2)  = 1;
    }
}

int minLen( )
{
    int j, k;
    for(j = 0; j < n; j++)dist[j] = g[0][j];
    dist[0] = 1;
    do{
        for(k = -1, j = 0 ; j < n; j++)  /* 找下一个最少上车次数的站*/
            if(dist[j] > 0 && (k == -1 || dist[j] < dist[k]))k = j;
        if (k < 0 || k == n-1)  break;
        dist[k] = -dist[k];  /* 设置 k 站已求得上车次数的标记 */
        for(j = 1; j < n; j++)  /* 调整经过 k 站能到达的其余各站的上车次数 */
            if ( (3)  && (dist[j] == 0 || -dist[k] + 1 < dist[j]))
                dist[j] =  (4) ;
    }while(1);
    j = dist[n-1];
    return  (5) ;
}
void main()
{
    int t;
    buildG();
```

```
    if((t = minLen()) < 0) printf ("无解!\n");
    else printf("从站 0 到站 %d 需换车 %d 次\n", n-1, t);
}
```

从下列的 3 道试题(试题五~试题七)中任选 1 道解答。如果解答的试题数超过 1 道，则题号小的 1 道解答有效

试题五(共 15 分)

阅读下列函数说明和 C 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【函数说明】

为网球比赛的选手安排比赛日程。设有 $n(n = 2^m)$位选手参加网球循环赛，循环赛共进行 n−1 天，每位选手要与其他 n−1 位选手赛一场，且每位选手每天赛一场，不轮空。

设 n 位选手被顺序编号为 1, 2, …, n，比赛的日程表是一个 n 行 n−1 列的表，第 i 行 j 列的内容是第 i 号选手第 j 天的比赛对手。用分治法设计日程表，就是从其中一半选手(2^{m-1}位)的比赛日程导出全体 2^m 选手的比赛日程。从众所周知的只有两位选手的比赛日程出发，反复这个过程，直至为 n 位选手安排好比赛日程为止。

如两位选手的比赛日程表如图 7-68 所示。

4 位选手的比赛日程表如图 7-69 所示。

	1
1	2
2	1

图 7-68　两位选手的比赛日程表

	1	2	3
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

图 7-69　4 位选手的比赛日程表

函数中使用的预定义符号如下。

```
#define M 64
int a[M+1][M];
```

【C 程序】

```
void main(){
    int twoml, twom, i, j, m, k;
    printf("指定 n(=2 的 k 次幂)位选手，请输入 k: \n");
    scanf("%d", &k);
    /*预设两位选手的比赛日程*/
    a[1][1] = 2;
    a[2][1] = 1;
    m = 1;
    twoml = 1;
    while( (1) ){
        m++;
```

```
            twoml += twoml;
            twom = twoml * 2;/*为 2^m 位选手安排比赛日程*/
            /*填日程表的左下角*/
            for(i = twoml + 1; (2) ; i++){
                for(j = 1; j <= twoml - 1; j++){
                    a[i][j] = a[i - twoml][j] + twoml;
                }
            }
            /*填日程表的右上角*/
            a[1][twoml] =  (3) ;/*填日程表右上角的第 1 列*/
            for(i = 2; i <= twoml; i++){
                a[i][twoml] = a[i - 1][twoml] + 1;
            }
            /*填日程表右上角的其他列,参照前一列填当前列*/
            for(j = twoml + 1; j < twom; j++){
                for(i = 1; i < twoml; i++){
                    a[i][j] =  (4) ;
                }
                a[twoml][j] = a[1][j - 1];
            }
            /*填日程表的右下角*/
            for(j = twoml; j < twom; j++){
                for(i = 1; i <= twoml; i++){
                    a[ (5) ][j] = i;
                }
            }
            /*输出日程表*/
            for(i = 1; i <= twom; i++){
                for(j = 1; j < twom; j++){
                    printf("%4d", a[i][j]);
                }
                printf("\n");
            }
            printf("\n");
        }
    }
```

试题六(共 15 分)

阅读以下说明和 C++代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

很多时候，希望某些类只有一个或有限的几个实例，典型解决方案是所谓单身(Singleton)模式。但在多线程情况下，Singleton 模式有可能出现问题，需要进行同步检查。如果对“检查 Singleton 对象是否已经创建”进行同步，则存在严重的瓶颈，所有的线程都必须等待检查对象是否存在。解决方式是一种称为 Double-Checked-Locking 的模式，其意图是将非必需的锁定优化掉，同步检查最多只发生一次，因此不会成为瓶颈。以下是 C++语言实现，能够正确编译通过。

【C++程序】

```
class USTax{
 (1) :
    USTax(){};//构造函数
public:
    static USTax* getInstance();
private:
    static USTax* instance;
};
 (2)  = NULL;
USTax* USTax::getInstance(){
    if(instance == NULL){
        //进行某种同步
        cout<<"实例暂时不存在"<<endl;
        if( (3) ){
            cout<<"实例不存在，创建实例..."<<endl;
            instance =  (4) ;
            cout<<"实例创建成功"<<endl;
        }
        else{
            cout<<"实例已被创建"<<endl;
        }
    }
    else{
        cout<<"实例已经存在"<<endl;
    }
    return  (5) ;
}
```

试题七(共 15 分)

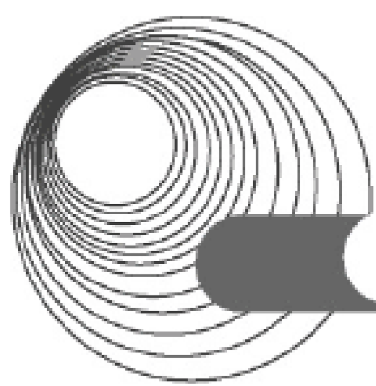
阅读以下函数说明和 Java 代码，将应填入(n)处的子句写在答题纸的对应栏内。

【说明】

很多时候，希望某些类只有一个或有限的几个实例，典型解决方案是所谓单身(Singleton)模式。但在多线程情况下，Singleton 模式有可能出现问题，需要进行同步检查。如果对“检查 Singleton 对象是否已经创建”进行同步，则存在严重的瓶颈，所有的线程都必须等待检查对象是否存在。解决方式是一种称为 Double-Checked-Locking 的模式，其意图是将非必需的锁定优化掉，同步检查最多只发生一次，因此不会成为瓶颈。以下是 Java 语言实现，能够正确编译通过。

【Java 程序】

```
public class USTax {
    private static USTax instance = null;
     (1)  USTax(){}
    private  (2)  static void doSync(){
        if(instance == null){
```




```
                System.out.println("实例不存在，创建实例...");
                instance = (3);
                System.out.println("实例创建成功");
            }else{
                System.out.println("实例已被创建");
            }
        }
        public static USTax getInstance(){
            if(instance == null){
                System.out.println("实例暂时不存在");
                (4);//同步控制
            }else{
                System.out.println("实例已经存在");
            }
            return (5);
        }
    }
```

7.2 样 卷 解 析

7.2.1 样卷一解析

软件设计师样卷一答案与解析

【试题一】

答案：

【问题 1】

0 层数据流图中的“试卷得分表”是局部文件，可不必画出。

【问题 2】

(1) 分类统计成绩中需要读入考生成绩，缺少从“考生名册”到“2.4 分类统计成绩”的数据流。

(2) “2.1 检查成绩表”缺少输出数据流“错误成绩表”。

(3) “2.2 审定合格者”缺少输入数据流“合格标准”。

【问题 3】

(1) 准考证号 + 姓名 + {课程名 + 成绩} + 合格/不合格标志。

(2) 报名号 + 姓名 + 通信地址 + 出生年份 + 文化程度 + 职业。

解析：

【问题 1】

“不必画出”是指在某层数据流图中只画流程图中各加工之间的公共数据文件，隐藏某加工的局部数据文件。这个规则只是为了使整个数据流图的层次结构更科学、更清晰，然而画出“不必画出的数据文件”对数据流图不会造成理解错误。在 0 层数据流图中有文

件“考生名册”和“试卷得分表”，其中“试卷得分表”是加工 2“统计成绩”的局部数据文件，所以不必画出。

【问题 2】

本问题是要指出图 7-3 中遗漏了哪些数据流，这需要从两个方面进行考虑。

一是父图与子图的平衡，即子图的输入/输出数据流与父图相应的加工的输入/输出数据必须一致。

二是针对每个加工至少要有一个输入和输出，反映此加工的数据来源和结果。

数据流图 7-3 是加工 2“统计成绩”的子图，为了发现图中遗漏的数据流，首先要观察 0 层数据流图中加工 2 的输入/输出数据流。在 0 层数据流图中，加工 2“统计成绩”有两个输入数据流“合格标准”和“成绩表”，4 个输出数据流“考生通知单”“统计分析表”、“难度分析表”和“错误成绩表”。

再看加工 2 子图中只有一个输入数据流“成绩表”，可见必然遗漏了一个输入数据流“合格标准”。根据题目说明提到的“对阅卷站送来的成绩表进行检查，并根据考试中心指定的合格标准审定合格者”，所以输入数据流“合格标准”应该是输入到加工 2.2“审定合格者”。

加工 2 子图中只有 3 个输出数据流“考生通知单”“统计分析表”和“难度分析表”，缺少数据流“错误成绩表”。加工 2.1“检查成绩表”的功能是检查成绩表是否合格，其中一个输出数据流是“正确成绩表”，自然另一个输出数据流应是“错误成绩表”。因此，第二个遗漏的数据流是加工 2.1“检查成绩表”的输出数据流“错误成绩表”。

根据题目中提到的“根据考生信息及考试成绩，按地区、年龄、文化程度和职业进行成绩分类统计及试题难度分析，产生统计分析表”这一说明，可以判断出加工 2.4“分类统计成绩”除了需要“试卷得分表”的输入外，还需要考生信息，也就是需要从文件“考生名册”的输入。

【问题 3】

根据题目说明中提到的“填写考生通知单(内容包含该考生的准考证号、姓名、各课程成绩及最终合格/不合格标志)，送给考生”可知，考生通知单应该包括考生的准考证号、姓名和最终合格/不合格标志，这种共同组成的含义由符号“+”来表示。同时因为考试可能由多门课程共同组成，所以，课程号和该课程的成绩也是必需的。其中的多门课程由符号“{…}”来表示重复。因此，考生通知单 = 准考证号 + 姓名 + {课程名 + 成绩} + 合格/不合格标志。

根据题目说明中提到的“对合格的报名表编好准考证号码后将准考证送给考生”，在 0 层数据流图中可以看到，加工 1“登记报名表”把考生信息写入文件“考生名册”中，可见“考生名册”中的数据除“准考证号”外均从合格的报名表中得到，因此“报名表”至少需要由报名号、姓名、通信地址、出生年份、文化程度和职业组成。由数据字典定义式表示为：报名表 = 报名号 + 姓名 + 通信地址 + 出生年份 + 文化程度 + 职业。

【试题二】

答案：

【问题 1】

Customer(<u>idno</u>, name, address, phone)

Account(<u>Ano</u>, balance, Bname)

    Bname reference Branch(Bname)

Deposit(<u>idno, Ano</u>)

    idno reference Customer(idno)

    Ano reference Account(Ano)

Branch(<u>Bname</u>, city)

Loan(<u>Lno</u>, Bname, amount)

    Bname reference Branch(Bname)

Borrow(<u>idno</u>, <u>Lno</u>)

    idno reference Customer(idno)

    Lno reference Loan(Lno)

【问题 2】

(1) Borrow.Lno =Loan.Lno。

(2) COUNT(distinct idno)>=2。

【问题 3】

这样的系统算不上分布式数据库系统。分布式数据库系统并不是简单地把集中式数据库系统安装在不同场地，用网络连接起来便实现了(这是分散的数据库系统)，而是具有自己的性质和特征。

分布式数据库系统具有以下特点。

(1) 数据的物理分布性。

(2) 数据的逻辑整体性。

(3) 数据的分布独立性。

(4) 场地的自治和协调。

(5) 数据的冗余及冗余透明性。

虽然上述银行的数据库系统具有性质(1)、(3)以及(4)和(5)的一部分，但关键是没有数据的逻辑整体性和不同场地之间的协调性等，而这恰恰是分布式数据库系统的关键所在。因此上述银行数据库系统算不上分布式数据库系统。

解析：

【问题 1】

将 E-R 模型转换为关系模式时，所需要遵循的转换规则如下。

(1) 每个实体集转换为一个关系。

(2) 一个一对一的联系可转换为一个关系模式，将两端关系的码及联系的属性均作为该关系的属性，任意一端的码作为该关系的码；也可将任意一端的码及联系的属性合并在另一端实体集所转换的关系模式中。

(3) 一个一对多联系可转换为一个关系模式，将两端关系的码及联系的属性均作为该关系的属性，“多”端的码作为该关系的码；也可将“一”端的码及联系的属性合并在“多”端实体集所转换的关系模式中。

(4) 一个多对多联系应转换为一个关系模式，两端的码及联系的属性为关系的属性，两端的码共同组合为该关系的码。

3 个或 3 个以上多对多的联系应转换为一个关系，各关系的码及联系的属性为关系的属性，各端的码共同组合为该关系的码。

本题的 E-R 图中有 4 个实体集、2 个多对多联系和 2 个一对多联系，根据上述 E-R 图转换关系模型的规则可以转换成 6 个关系。

4 个实体集转换为 4 个关系(即 Customer、Account、Branch 和 Loan)，而一对多联系 B-L 和 B-A 则是将“一”端(即关系 Branch)的码 Bname 加入到“多”端所转换的关系(即 Account 和 Loan)中。此 4 个关系分别为

Customer(idno, name, address, phone)

Account(Ano, balance, Bname)

Branch(Bname, city, assets)

Loan(Lno, Bname, amount)

4 个关系中，Account 和 Loan 的属性 Bname 均参照 Branch 的码 Bname，为外码。

2 个多对多联系转换为 2 个关系，两端的码及联系的属性为关系的属性，两端的码共同组合为该关系的码。此 2 个关系分别为

Deposit(idno, Ano)

Borrow(idno, Lno)

其中的 idno、Ano 和 Lno 分别参照 Customer 的 idno、Account 的 Ano 和 Loan 的 Lno。

【问题 2】

本问题是要查询在该银行中一笔贷款贷给多个(至少两个)客户的所有贷款号和发放贷款的支行名称。Borrow 表中记录着各贷款号和该贷款的客户，Loan 表中记录着各贷款号和发放该贷款的支行，要完成题目查询必须将 Borrow 和 Loan 联系起来，即需要两者的贷款号相等。所以空(1)处应为 Borrow.Lno =Loan.Lno。

“一笔贷款贷给多个客户”则需要按贷款号进行分组，只有客户个数至少为两个的组才是满足查询要求的分组。对于分组的条件应该添加在 HAVING 子句中，个数的统计需利用 COUNT(idno)函数，因此空(2)处应为 COUNT(distinct idno)>=2。

【问题 3】

本问题主要考查分布式数据库系统的必备条件。

【试题三】

答案：

【问题 1】

(1) Return(还书)。　(2) Outdate(超期)。

(3) Reader Register(读者注册)。　(4) Book Cancel Register(图书注销)。

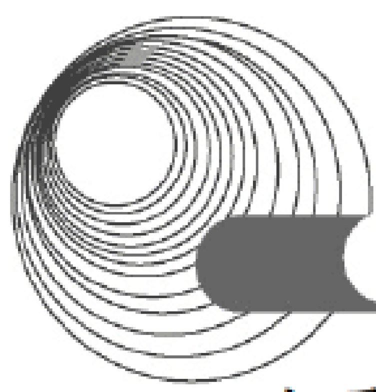
【问题 2】

(5) 0..1。　(6) 0..1。

【问题 3】

(1) 全局的约束应用于一个链接角色，指定相应的实例是可见的，因为它是一个全局量。通过系统都知道的全局名就可以得到该实例。

(2) 局部的约束应用于一个链接角色，指定相应的实例是可见的，因为它是操作的局部



变量。

(3) 参数应用于一个链接角色，指定相应的实例是可见的，因为它是某个操作的参数。

(4) 自我应用于一个链接角色，指定对象可以向它自己发送消息。

(5) 投票应用于消息，限制一组返回的消息。投票约束指定从这组消息中以大多数投票的方式选出返回的值。

(6) 广播应用于消息，指定不能以某种顺序发出该消息。

(7) 创建影响对象生命，将在交互的执行中创建对象。

(8) 注销影响对象生命周期，对象将在一个交互的执行中被注销。

(9) 临时是影响对象生命周期的一个约束。临时的对象的创建和注销是在一个操作的同一执行中，它是创建和注销约束的结合。

解析：

【问题 1】

仔细阅读题目可以发现，“图书馆不定期地购买并注册新图书供读者借阅，也可将报废的旧书注销以停止借阅”这句需求的前半部分已经在用例图中显示出来，后半部分没有用例图说明，同时空(4)处的前件是 Book Register(图书注册)，可以得出空(4)处是 Book Cancel Register(图书注销)。同样道理，从题目说明中“每个读者在借阅之前需注册姓名、性别、地址、E-mail 等内容”，得到空(3)处是 Reader Register(读者注册)。从题目说明中“一本书超期一天罚款 0.1 元。若一本书超期 3 个月不归还，则发布通告”，得到空(2)处是 Outdate(超期)，空(1)处是 Return(还书)。

【问题 2】

在 UML 中重复度表示为一个整数范围 n..m，整数 n 定义所连接的最少对象的数目，而 m 则为最多对象的数目(当不知道确切的最大数时，最大数用*号表示)。例如：

0..1 表示零到 1 个对象；

0..*或者*表示零到多个对象；

5..17 表示 5~17 个对象；

2 表示 2 个对象。

【问题 3】

题目中的 9 个约束应用于交互。可以用顺序图从时间的角度来看交互，也可以用协作图从空间的角度来看交互，或者还可以用活动图从工作的角度来看交互。在所有类型的交互图中(顺序图、协作图和活动图)，消息、对象和链接是最重要的概念。对象是通过链接(而不是关联)连接起来的，每个链接都有两个链接角色，即关联角色的实例。当对象进行交互时，它们扮演角色(链接的角色)并且通过链接相互发送消息。

【试题四】

答案：

(1) j--。　　(2) i++。　　(3) A[i]←pivot 或 [j]←pivot。

(4) A，L，k-1 或 A，L，k。　　(5) A，k+I，H 或 A，k，H。

解析：

本题目考查快速排序算法。快速排序采用了一种分治的策略，通常称为分治法。其基

本思想是：将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

快速排序的具体过程为：第一步，在待排序的 n 个记录中任取一个记录，以该记录的排序码为基准，将所有记录分成 2 组，第一组各记录的排序码都小于等于该排序码，第二组各记录的排序码都大于该排序码，并把该记录排在这 2 组中间，这个过程称为一次划分。第二步，采用同样的方法，对划分出来的 2 组元素分别进行快速排序，直到所有记录都排到相应的位置为止。

在进行一次划分时，若选定以第一个元素为基准，就可将第一个元素备份在变量 pivot 中，如图 7-7 中的第①步所示。如此一来，基准元素在数组中占据的位置就空闲出来了，因此下一步就从后向前扫描，如图 7-7 中的第②步所示，直至找到一个比基准元素小的元素时为止，将其前移，如图 7-7 中的第③步所示。然后再从前向后扫描，如图 7-7 中的第④步所示，直至找到一个比基准元素大的元素时为止，将其后移，如图 7-7 中的第⑤步所示。这样，从后向前扫描和从前向后扫描交替进行，直到扫描到同一个位置为止，如图 7-7 中的第⑥步所示。

由题目中给出的流程图可知，以第一个元素作为基准数，并将 A[low]备份至 pivot，i 用于从前向后扫描的位置指示器，其初值为 low，j 用于从后往前扫描的位置指示器，其初值为 high。当 $i<j$ 时进行循环。

(1) 从后向前扫描数组 A，在 $i<j$ 的情况下，如果被扫描的元素 $A[j]>pivot$，就继续向前扫描(j--)；如果被扫描的元素 $A[j]<pivot$ 就停止扫描，并将此元素的值赋给目前空闲着的 A[i]。

(2) 这时，再从前向后扫描，在 $i<j$ 的情况下，如果被扫描的元素 $A[j]<pivot$，就继续向后扫描(i++)；如果被扫描的元素 $A[j]>pivot$ 就停止扫描，并将此元素的值赋给目前空闲着的 A[j]。

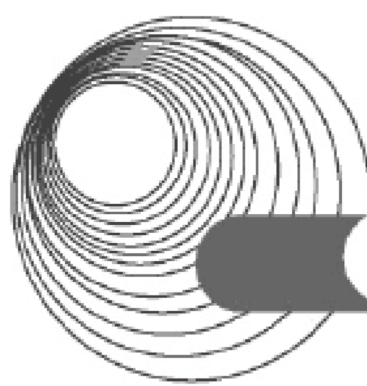
(3) 这时又接第(1)步，直到 $i>j$ 时退出循环。退出循环时，将 pivot 赋给当前的 A[i] (A[i] ←pivot)。

递归函数的目的是执行一系列调用，直至到达某一点时递归终止。为了保证递归函数正常执行，应该遵守下面的规则。

(1) 每当一个递归函数被调用时，程序首先应该检查一些基本的条件是否满足。例如，某个参数的值等于零，如果是这种情形，函数应停止递归。

(2) 每次当函数被递归调用时，传递给函数一个或多个参数，应该以某种方式变得“更简单”，即这些参数应该逐渐靠近上述基本条件。例如，一个正整数在每次递归调用时会逐渐变小，以至最终其值达到零。

本题中，递归函数 sort(int A[]，int L，int H)有 3 个参数，分别表示数组 A 及其下界和上界。根据流程图可知，这里的 L 相当于流程图中的 i，这里的 H 相当于流程图中的 j。因为 p()返回基准数所在数组 A 中的下标，也就是流程图中最后的“A[i]←pivot”中的 i。根据快速排序算法，在第一趟排序后找出了基准数所在数组 A 中的下标，然后以该基准数为界(基准数在数组中的下标为 k)，把数组 A 分成 2 组，分别是 A[L,…, k−1]和 A[k+1, …, H]，最后对这 2 组中的元素再使用同样的方法进行快速排序。



【试题五】

答案：

(1) NULL。 (2) n。 (3) n−1。 (4) newNode→next。
(5) head。 (6) n。 (7) n。 (8) No++。
(9) head->next。 (10) head

解析：

如果掌握了链表的创建、遍历和删除的方法，解决本题应该并不困难。要显示链表各节点的数据，就是要把各节点找到，然后把该节点的每一个 x、y 坐标打印出来。不过，本链表与普通链表也有不同的地方，即该链表的节点数据是指针。要在链表节点中存入数据，必须先动态分配存储数据的内存单元；要删除链表中的各个节点，必须先释放节点数据的内存单元，否则会造成内存泄漏。

【试题六】

答案：

(1) Object。 (2) virtual。 (3) Iterator<Object>。
(4) &BookShelf。 (5) it->hasNext()。

解析：

Iterator 是模板类，空(1)处应该填某个类名，其方法 next()的返回类型是 Object，而 Object 没有定义，故空(1)处应填 Object。

由 next()方法末尾的“=0”可知，该方法为纯虚函数，故空(2)处应填 virtual。

BookShelfIterator 类继承自 Iterator 类，要注意的是模板类基类的写法，空(3)处应填 Iterator<Object>。

根据构造函数 BookShelfIterator(BookShelf *bookShelf)可得，空(4)处应填&BookShelf，注意地址符，因形参是 BookShelf 指针。

while 循环是遍历书架，输出书名，循环条件是“还有下一记录(书)”，故空(5)处应填 it->hasNext()。注意指针写法。

【试题七】

答案：

(1) implements。 (2) this。 (3) iterator()。
(4) it->hasNext()。 (5) next()。

解析：

Iterator 是接口类，空(1)处应填 implements。根据构造函数 BookShelfIterator(BookShelf bookShelf)可得，空(2)处应填 this，即自身引用。

空(3)处是取得迭代器实例，BookShelf 类中的方法 iterator()是返回 Iterator 接口，故空(3)处应填 iterator()。

while 循环是遍历书架，输出书名，循环条件是“还有下一记录(书)”，故空(4)处应填 it->hasNext()。注意指针写法。

空(5)处是取得书实例，BookShelf 类中的方法 next()是返回 Object 类实例，取得下一本书，故空(5)处应填 next()。

7.2.2　样卷二解析

软件设计师样卷二答案与解析

【试题一】

答案：

【问题 1】

(1) 名称：退货单。起点：物料检验。终点：采购员。

(2) 名称：缺货单。起点：领料单检验。终点：领料人。

【问题 2】

(1) 领料单的属性：物料代码、数量、日期、领料人、仓库保管员。

(2) 入库申请单的属性：物料代码、数量、供货方、日期、采购员。

【问题 3】

库存超限报警、库存不足报警。

【问题 4】

采购计划单、入库单、供货方档案、出库单、物料主文件、领料计划单。

解析：

本题和第 1 章中的几道例题略有不同，题目给出的不是分层数据流图，而是一张系统的详细数据流图。同时题目不仅考查了数据流图的基本知识和数据字典条目，还考查了用面向对象方法设计系统的有关知识。

【问题 1】

此问题要求考生找出数据流图中缺少的数据流。由于该题没有其他任何图可以对比和参考，所以解题的关键就是仔细阅读说明，将文字与图一一对照，找出问题所在。

本题的说明部分对物料出入库管理工作流程的阐述非常清晰，所以只要逐句分析，并在数据流图中找到相应的数据流，若没有，则为缺少的数据流。例如，在出库工作流程中，“(1)领料人提交领料单”对应数据流图中一条从“领料人”到“领料单检验”的“领料单”数据流。逐步检查后发现，“(5)若没有足够的库存，仓库保管员向领料人发缺货单”这句话在数据流图中找不到相应的数据流，所以缺少一条从“领料单检验”到“领料人”的“缺货单”数据流。同样道理，在入库工作流程中，无法找到与“(5)如果物料或供货方不合格，则向采购员发出退货单”相对应的数据流，所以缺少一条从“物料检验”到“采购员”的“退货单”数据流。

【问题 2】

此问题要求找出“领料单”和“入库申请单”的数据字典条目，可以根据题目的说明部分来解答此问题。

首先在说明中分别找到与之相关的描述。在出库工作流程中，每句话都提到了领料单，根据其中的(1)和(4)可知领料单应包含属性“物料代码”；根据(4)可知领料单应包含属性“数量”和“日期”；由(1)、(3)、(5)都可知领料单应包含属性“领料人”；由(6)可推出领料单还应包含属性“仓库保管员”。所以领料单应该包含属性：物料代码、数量、日期、领料人、仓库保管员。

同理，在入库工作流程中可以找到与“入库申请单”有关的描述。由(1)可知，入库申请单应包含属性“采购员”和“物料代码”；由(2)和(5)可知，入库申请单应包含属性“日期”和“供货方”；根据(2)和(6)可推出领料单还应包含属性“数量”。

【问题 3】

此问题是一个典型的进销存问题。对于一功能完善的库存管理系统，应具有查询、统计、报表输出和库存报警功能。分析题目可知，在出库时应有库存不足报警，在入库时应用库存超限报警。

【问题 4】

此问题要求在采购计划单、物料主文件、领料单、出库单、供货方档案、入库申请单、检验单、入库单和领料计划单这 9 个类中找出需要持久存储的类。在数据流图中，可以看到“采购计划单、入库单、供货方档案、出库单、物料主文件、领料计划单”表示为文件，可见这些文件中存储的信息需要被永久保存，所以这些文件需要映射到关系数据库模式中。而领料单、入库申请单、检验单只在数据流中出现，可见这些只是暂时的凭证，不需要永久保存。

【试题二】

答案：

【问题 1】

设计一中的关系模式 Invoice 最高满足第一范式。

根据题意可得出以下函数依赖：

Ino -> Sno, Cno, Idate

而关系模式 Invoice 的主码是 Ino 和 Mno。非主属性 Sno、Cno 和 Idate 并非完全依赖于主码，因此关系模式 Invoice 不满足第二范式，最高满足第一范式。

设计二更加合理。因为设计二解决了设计一中由于非主属性不完全依赖于主码而造成的数据冗余等问题。

【问题 2】

(1) AS。　(2) SUM(amount)。　(3) SUM(unitprice*amount)。

(4) Invoice.Ino=Invoicedetail.Ino。　(5) Invoice.Ino,Idate 或 Invoicedetail.Ino,Idate。

【问题 3】

(1) A 或 AS A。　(2) NOT EXISTS。　(3) *。

【问题 4】

有必要。Merchandise 中由属性 price 表示的是商品的当前价格，超市中的价格是有可能变动的；而关系 Invoicedetail 中的属性 unitprice 表示的是在开具发票时该商品的单价。

解析：

【问题 1】

关系数据库设计的方法之一就是设计满足适当范式的模式，通常可以通过判断分解后的模式达到几范式来评价规范化的程度。

(1) 1NF(第一范式)：若关系模式 R 的每一个分量是不可再分的数据项，则关系模式 R 属于第一范式。

(2) 2NF(第二范式)：若关系模式 R∈1NF，且每一个非主属性完全依赖于码，则关系模式属于第二范式。当 1NF 消除了非主属性对码的部分函数依赖，则称为 2NF。

(3) 3NF(第三范式)：若关系模式 R(U,F)中不存在这样的码 X、属性组 Y 及非主属性 Z(Z⊆Y)使得 X→Y，(Y→X)Y→Z 成立，则关系模式属于 3NF。即当 2NF 消除了非主属性对码的传递函数依赖，则称为 3NF。

(4) BCNF(巴克斯范式)：若关系模式 R∈1NF，当 X→Y 且 Y⊆X 时，X 必含有码，则关系模式属于 BCNF。即当 3NF 消除了主属性对码的部分和传递依赖，则称为 BCNF。

(5) 4NF(第四范式)：关系模式 R∈1NF，若对于 R 的每个非平凡多值依赖 X→→Y 且 Y⊆X 时，X 必含有码，则关系模式属于 4NF。4NF 是限制关系模式的属性间不允许有非平凡且非函数依赖的多值依赖。

设计一中根据题意可得出以下函数依赖：

Ino → Sno, Cno, Idate

而关系模式 Invoice 的主码是 Ino 和 Mno。非主属性 Sno、Cno 和 Idate 并非完全依赖于主码，因此关系模式 Invoice 不满足第二范式，最高满足第一范式。

关系模式 Invoice 设计的不合理在于该关系模式中将发票的单值属性(发票号码 <u>Ino</u>，交易日期 Idate，顾客代码 Cno，收银员代码 Sno)和多值属性(商品代码 <u>Mno</u>，单价 unitprice，数量 amount)混合在一个关系模式中，造成关系模式 Invoice 的冗余异常、修改异常和删除异常。而设计二则将设计一中的关系模式 Invoice 分解，使得发票的单值属性和多值属性分开，避免了异常。因此，设计二明显比设计一要好。

【问题 2】

本问题是要建立 2005 年 1 月期间每张发票的发票号、交易日期、交易商品件数和交易总金额的视图。

首先建立视图的格式为 CREATE VIEW <视图名> AS <视图定义>，因此空(1)处的答案为 AS。

本查询是从 Invoice 和 Invoicedetail 两个关系模式中查询，两关系模式的连接条件是两关系模式的 Ino 相等，因此空(4)处的答案是 Invoice.Ino=Invoicedetail.Ino。

统计每张发票的信息需要按发票将数据分组，也就是按发票号 Ino 分组，但因为查询关系模式 Invoice 和 Invoicedetail 都有属性 Ino，为了避免二义性，所以分组属性是 Invoice.Ino 或 Invoicedetail.Ino。因为在包含聚合运算的 Select 子句中，只有在 Group By 子句中出现的属性才能在 Select 子句中以非聚合形式出现，而 Select 子句中有非聚合形式的属性 Idate 出现，所以空(5)处的答案是 Invoice.Ino,Idate 或 Invoicedetail.Ino,Idate。

需要查询的是每张发票的交易商品件数和交易总金额。交易商品件数是发票商品数量的总和，因此空(2)处的答案是 SUM(amount)。交易总金额是每条交易商品明细中每条记录商品金额的总和，每条记录商品金额是 unitprice*amount，因此空(3)处的答案是 SUM(unitprice*amount)。

【问题 3】

本问题是要查询从未售出的商品信息。

SQL 语句中有两种格式为表或视图取别名：“表名 AS 别名”或“表名 别名”。由题中可以看出 Merchandise 的别名是 A，因此空(1)处的答案是 A 或者 AS A。

要查询“从未出售”的商品，也就是要查询的商品在交易记录中不存在，因此空(2)处的答案是 NOT EXISTS。

【问题 4】

Merchandise 中由属性 price 表示的是商品的当前价格，超市中的价格是有可能变动的；而 Invoicedetail 中的属性 unitprice 表示的是在开具发票时该商品的单价。如果缺少其中任意一个，将导致商品单价不能进行调整，否则当商品的单价发生变化时，销售历史中的商品价格就随着发生变化。

【试题三】

答案：

【问题 1】

(1) cancel register (注销旧车的注册)。

(2) register(车辆注册)。

(3) return(归还)。

(4) lost report(遗失报告)。

【问题 2】

(5) cancel register(注销用户的注册)。

(6) borrow/n+1(借车/拥有车的数量 + 1)。

(7) penalty and n=0(罚款并且拥有车的数量为 0)。

(8) 3。

【问题 3】

活动图(Activity Diagram)显示动作及其结果。活动图着重描述操作(方法)实现中所完成的工作以及用例实例或对象中的活动。活动图是状态图的一个变种，与状态图的目的有一些小的差别，活动图的主要目的是描述动作(执行的工作和活动)及对象状态改变的结果。当状态中的动作被执行(不像正常的状态图，它不需指定任何事件)时，活动图中的状态(称为动作状态)直接转移到下一个阶段。活动图和状态图的另一个区别是活动图中的动作可以放在泳道中。泳道聚合一组活动，并指定负责人和所属组织。活动图是另一种描述交互的方式，描述采取何种动作、做什么(对象状态改变)、何时发生(动作序列)以及在何处发生(泳道)。

活动图可以用作下述目的。

(1) 描述一个操作执行过程中(操作实现的实例化)所完成的工作(动作)。这是活动图最常见的用途。

(2) 描述对象内部的工作。

(3) 显示如何执行一组相关的动作，以及这些动作如何影响它们周围的对象。

(4) 显示用例的实例如何执行动作以及如何改变对象状态。

解析：

【问题 1】

根据题意可以分析出车辆的状态和事件。例如，根据“车库不定期地购买并注册新车供用户借用，也可将报废的旧车注销以停止租用”，可以得出空(1)处、空(2)处分别是注销旧车、车辆注册。空(3)处可以从在库状态和在借状态的合理推断，得出从在借到在库只有一种事件——“归还”。从在借状态到终结状态，也只有一种可能性，那就是遗失。

【问题 2】

根据题意“注销用户之前，该用户必须归还所有借的车，或者报失并接受罚款”，得出从“No Car”到终结状态的事件空(5)是 cancel register(注销用户的注册)。根据从“No Car”到“Has Car”的事件空(6)是 borrow(借车)，同时已知用户可以借多辆车，当前拥有车 n 辆，所以需要 n + 1。根据“若用户借的车丢失，在罚款处理之前不能借车”可知空(7)处是 penalty(罚款)，同时状态从“Has Car”到达“No Car”说明 n=0。根据“每个用户最多可同时借 3 辆车”，可以得出空(8)处为 3。

【问题 3】略。

【试题四】

答案：

(1) G.vexnum。　(2) fix_mfset(F, LocateVex(e.vex2))。

(3) !=。　(4) k++。　(5) i++。

解析：

本题考查的是克鲁斯卡尔(Kruskal)算法。理解该算法的关键在于：由于生成树上不允许有回路，因此并非每一条当前权值最小的边都可选。例如，对图 7-70 所示连通网 G5，在依次选中了(e,f)、(b,c)、(e,d)和(f,g)的 4 条边之后，权值最小边为(g,d)。由于 g 和 d 已经连通，若加上(g,d)这条边将使生成树上产生回路，显然这条边不可取。同理边(f,d)也不可取，之后则依次取(a,g)和(a,b)两条边加入到生成树。

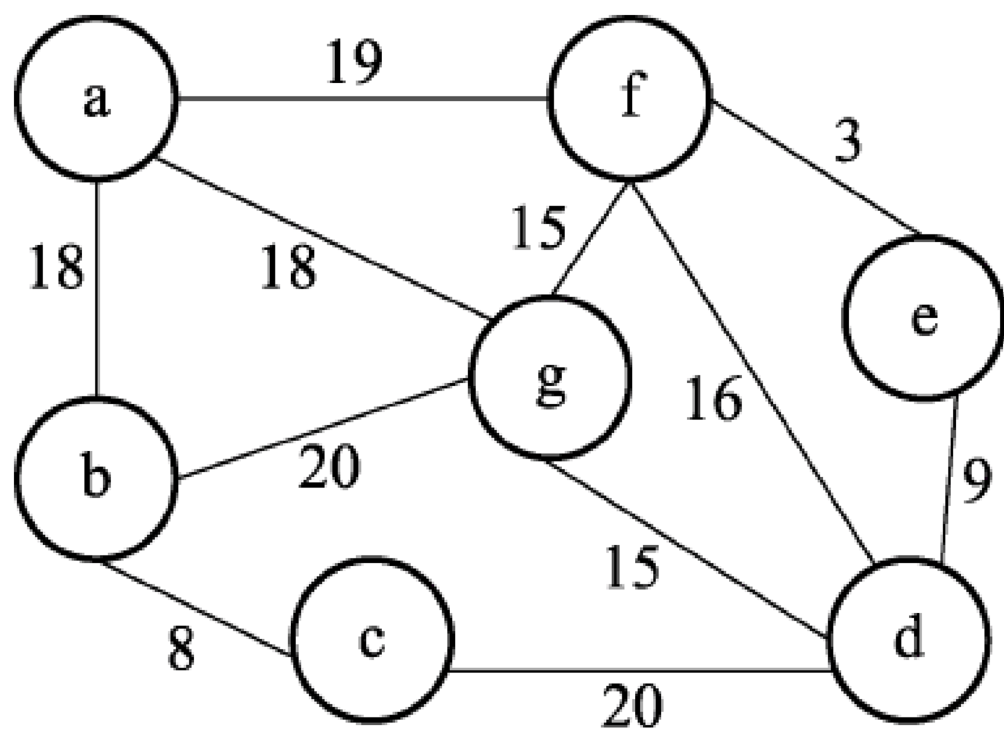


图 7-70　连通网 G5

那么在算法中如何判别当前权值最小边的两个顶点之间是否已经连通？从生成树的构造过程可见，初始态为 n 个顶点分属 n 棵树，互不连通，每加入一条边，就将两棵树合并为一棵树，在同一棵树上的两个顶点之间自然相连通。由此判别当前权值最小边是否可取只要判别它的两个顶点是否在同一棵树上即可。

【试题五】

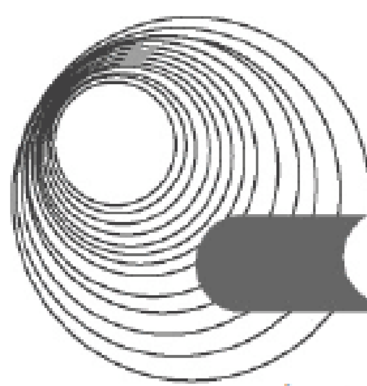
答案：

(1) vector<string>。　(2) Builder*。(3) items.clear()。(4) Builder。(5) construct()。

解析：

首先来看空(2)处，由名字可猜想 builder 是一个 Builder 类对象，由构造函数中的语句 this->builder = builder 及形参声明 Builder *builder，可判知空(2)处应填 Builder*。

由注释可知空(3)处是用来清空 items 向量的，items 是一个 vector<string>对象，此处并



未提供清空方法，应该是调用库函数，故应调用 items.clear()。

现在来看空(1)处，由语句 builder->makeItems(items)及 vector<string> items 可知，空(1)处应填 vector<string>。

继续看空(4)处，由类图知，TextBuilder 是 Builder 的子类，因此此处应填 Builder，声明继承关系。

空(5)处是真正进行文件的构造，应填 construct()。事实上，Director 类仅提供了该方法，自然是调用该方法。

【试题六】

答案：

(1) abstract。(2) Builder。(3) extends。(4) items.length。(5) construct()。

解析：

Builder 类含有多个 abstract 方法，故应声明为 abstract，空(1)处应填 abstract。

由构造函数中的语句 this.builder = builder 及形参声明 Builder builder，可判知空(2)处应填 Builder。

由类图知，TextBuilder 是 Builder 的子类，因此空(3)处应填 extends，声明继承关系。

空(4)处的 for 循环是将 items 中的对象添加到 buffer 中，空(4)处是循环终止条件，即下标达到 items 的长度，故应填 items.length。

空(5)处是真正进行文件的构造，应填 construct()。事实上，Director 类仅提供了该方法，自然是调用该方法。

【试题七】

答案：

(1) p == NULL。(2) pC->next = C->head->next。

(3) tp = NULL。(4) Del(C, tp)。(5) break。

解析：

根据注释，Del 函数当 p 是空指针时，删除头节点，因此易知空(1)处应填 p == NULL。

空(2)处是插入头节点的特殊情况，应填 pC->next = C->head->next。

由注释可知空(4)处是删除操作，需调用 Del 函数，进一步确定实参。此处需要删除节点 t，结合 Del 函数的说明，实参 p 需要空指针(NULL)或 t 节点的前驱指针，由 if(t->e > pC->e){tp = t; t = t->next;}可知，tp 正是 t 节点的前驱指针，因此应填 Del(C, tp)。由此也可以确定空(3)处，当需要删除头节点时，tp 应是空指针，因此空(3)处应填 tp = NULL。

至于空(5)处，应填 break。此时节点 p 已正确处理完毕，应该结束 while 循环，而且必须终止，否则 while 循环结束后 t 值为 NULL，将会错误地执行插入操作。

7.2.3 样卷三解析

软件设计师样卷三答案与解析

【试题一】

答案：

【问题 1】

(1) NOT IN (2) Rno。 (3) COUNT(DISTINCT Rno)。

(4) Bdate='2005-01-01'。 (5) EXISTS。 (6) Borrow.Bno=Books.Bno。

【问题 2】

不允许。表 Borrow 中，借书证号 Rno、管理员工作证号 Ano 和图书书号 Bno 共同组成主码，因此不允许有两个在主码上完全相同的元组(行)存在，也就是不允许同一读者从同一管理员处多次借阅同一本书。

解析：

【问题 1】

对于程序 1，从给定的 SQL 语句中可以看出，子查询中是查询所有借阅过编号为 111111 图书的所有读者。而题目要求是查询“没有借阅过编号为 111111 图书的所有读者名单”，也就是从读者 Readers 关系中查询不在子查询中出现的那些读者。所以空(1)处的答案是 NOT IN。SQL 语句中是 Readers 关系中的 Rno 与子查询结果进行比较，所以空(2)处的答案也应该是 Rno。

程序 2 要求查询人数，自然需要利用统计函数 COUNT。因为 2005 年 1 月 1 日同一读者可能借多本书，也就是可能在 Borrow 关系中出现多次，所以需要加上 DISTINCT 表示不计重复值。因此空(3)处的答案是 COUNT(DISTINCT Rno)。查询的条件自然是借阅时间是 2005 年 1 月 1 日，所以空(4)处的答案是 Bdate='2005-01-01'。

程序 3 是要查“借书证号为 123456 的读者所借过的所有图书”，也就是从 Books 关系中查询出图书信息，这些图书被借书证号为 123456 的读者借阅的记录在 Borrow 关系中存在。因此空(5)处的答案是 EXISTS，表示存在。空(6)处的答案是 Borrow.Bno=Books.Bno。

【问题 2】

主码是唯一识别表中记录的属性。在一个表中，不允许有两个在主码上完全相同的元组(行)存在。表 Borrow 中，借书证号 Rno、管理员工作证号 Ano 和图书书号 Bno 共同组成主码，所以不允许有任意两行在这 3 个属性上的值都完全相同，也就是不允许同一读者从同一管理员处多次借阅同一本书。

【试题二】

答案：

【问题 1】

医院收费系统的 0 层图中的“处方记录”可以不必画出。

【问题 2】

(1) 从“病人基本情况”到“3.1 检查处方单”的数据流。

(2) 从“3.2 记录处方”到“处方记录”的数据流。

(3) 从“定价表”到“3.3 制作收据”的数据流。

(4) 从“3.3 制作收据”到“收费记录”的数据流。

【问题3】

(1) “1.1 检查病人信息”的“不合格病人信息”输出数据流。

(2) “1.2 计算收费”的“收据”输出数据流。

解析：

【问题1】

0层图中的“处方记录”是加工3“处方收费”的局部数据文件，所以不必画出。

【问题2】

根据说明“系统首先根据病人基本情况检查处方单中病历号是否正确”，因此，在加工3.1“检查处方单”中，需读入病人基本情况，所以缺少从“病人基本情况”到“3.1 检查处方单”的数据流。然后系统“记录合格的处方单”，所以加工3.2“记录处方”中需将处方的内容记录到文件“处方记录”中，因此缺少从“3.2 记录处方”到“处方记录”的数据流。加工3.3“制作收据”中需根据文件“定价表”的各项目或药品的价格来计算所需收取的费用，因此图中还缺少从“定价表”到“3.3 制作收据”的数据流。最后收费的记录需写入文件“收费记录”中，所以缺少的第四条数据流是从“3.3 制作收据”到“收费记录”的数据流。

【问题3】

找出缺少的数据流的一个关键是父图与子图的平衡，即子图的输入/输出数据流与父图相应的加工的输入/输出数据必须一致。

从0层图中可以看到，对于加工1“病历收费”有输入数据流“病人信息”，输出数据流“不合格病人信息”“病历”和“收据”。而加工1子图中却只有“病人信息”和“病历”，所以一定缺少两条输出数据流“不合格病人信息”和“收据”。病人信息是否合格是在加工1.1“检查病人信息”中处理，因此加工1.1除一条输出数据流“合格病人信息”外，还缺少一条输出数据流“不合格病人信息”。对合格的病人信息，加工1.2“计算收费”处理后，理应提供收据给病人，所以另一条缺少的数据流是“1.2 计算费用”的“收据”输出数据流。

【试题三】

答案：

【问题1】

用例具有以下特征。

- 用例总是由角色初始化。
- 用例为角色提供值。
- 用例具有完全性，即不管其内部是如何实现的，只有最终产生了返回角色的结果，用例的执行才能完毕。

(1) 登记成绩。 (2) 查询成绩单。 (3) 关闭注册。

【问题2】

(4) create schedule()。 (5) display blank schedule()。 (6) get course offerings()。

(7) create with offerings()。　(8) add schedule(schedule)。

【问题 3】

- 用例图定义了系统的功能需求，它完全是从系统的外部观看系统功能，并不描述系统内部对功能的具体实现。在用例图中，角色代表触发系统功能的用户或其他系统，用例代表具体的功能描述。
- 类图描述系统的静态结构，表示系统中的类及类与类之间的关系。
- 对象图描述了一组对象及其关系，表示类的对象实例。
- 状态图表示一个状态机，强调对象行为的事件顺序。
- 时序图和协作图均表示一组对象之间的动态协作关系，其中时序图反映对象之间发送消息的时间顺序，协作图反映收发消息的对象的结构组织。时序图和协作图是同构的，即两者之间可以相互转换。
- 活动图反映系统中从一个活动到另一个活动的流程，强调对象间的控制流程。
- 组件图描述组件及其关系，表示系统的静态实现视图。
- 分布图反映了系统中软件和硬件的物理架构，表示系统运行时的处理节点以及节点中组件的配置。

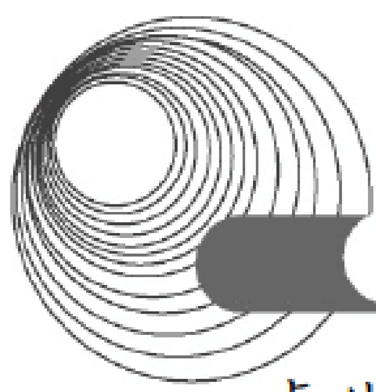
解析：

用例描述了它所代表的功能的各个方面，即包含了用例执行期间可能发生的种种情况。用例和角色之间具有“关联”的连接关系，表示什么角色与该用例进行通信。在 UML 语言中，用例用一个椭圆图形和名称表示。

在本题中，通过题目说明可以识别以下用例。

- 与教师有关的用例。
  选择课程：选择所教的课程，并获得学生名册。
  登记成绩：在学期结束时，提交学生的课程成绩。
- 与学生有关的用例。
  注册课程：在学期开始进行选课注册，允许在一段时间内更改或删除，课程目录系统提供当前学期的所有可选课程列表。
  查看成绩单：学生可以查看以前学期的电子成绩单。
- 与注册管理员有关的用例。
  维护课程信息：在系统中增加、修改和删除课程信息。
  维护学生信息：在系统中增加、修改和删除学生信息。
  维护教师信息：在系统中增加、修改和删除教师信息。
  关闭注册：删除少于 3 人的课程，并由付费系统通知学生缴费。
- 与安全性要求有关的用例。
  登录：使用此系统的人员需要进行登录，以验证其身份和权限。

发现和定义对象类应以问题域和系统责任为出发点，正确地运用抽象原则，尽可能全面地发现对象的因素，并对其进行检查和整理，最终得到系统的对象类。可以在用例模型的基础上，通过识别实体类、边界类和控制类，从而发现和定义系统中的对象类。识别上述对象类之后，通过建立交互图，将用例的行为分布到这些对象类中。时序图表示完成某项行为的对象类和这些对象类之间传递消息的时间顺序，其中，对象生命线是一条垂直的



虚线，表示对象存在的时间；控制焦点是一个细长的矩形，表示对象执行一个所经历的时间段；消息是对象之间的一条水平箭头线，表示对象之间的通信。协作图包含一组对象和以消息交换为纽带的关联，用于描述系统的行为是如何由系统的成分合作实现的。

【试题四】

答案：

(1) A.length。 (2) j++。 (3) O (Min(A.length, B.length))。

(4) m。 (5) p→next。 (6) p。 (7) L->next。

(8) q->next。 (9) ha。 (10) O(ListLength(L))。

解析：

函数 1 中，算法要求对两个顺序表进行“比较”，是一种“引用型”操作，因此在算法中不应该破坏已知表。按题目中的规定，只有在两个表的长度相等，且每个对应元素都相同时才相等；否则两个顺序表的大小主要取决于两表中除去最大公共前缀后的第一个元素。因此，比较两表的大小不应该先比较它们的长度，而应该设一个下标变量 j 同时控制两个表，即对两表中“位序相同”的元素进行比较。

上述算法中只有一个 while 循环，它的执行次数依赖于待比较的顺序表的表长，因此，算法的时间复杂度为 O(Min(A.length, B.length))。

函数 2 中，因为对链表来说，“插入”和“删除”仅需修改指针即可完成，并且由于前 m 个元素之间和后 n 个元素之间的链接关系分别都不需要改变，则算法的实际操作如下。

(1) 从链表中删除 $(a_1,a_2,\cdots,a_m)$。

(2) 将 $(b_1,b_2,\cdots,b_n)$ 链接到头节点之后。

(3) 将 $(a_1,a_2,\cdots,a_m)$ 链接到 b_n 之后。

算法的时间复杂度为 O(ListLength(L))。

【试题五】

答案：

【问题 1】

(1) 1。 (2) col。 (3) row。 (4) 2。 (5) col。 (6) row。 (7) k。

【问题 2】

判断条件 1：(b)。

判断条件 2：(e)。

判断条件 3：(f)。

解析：

【问题 1】

本问题中算法的功能是检查文本文件中的括号(圆括号、方括号、花括号)是否匹配。

从提示信息可以看出该算法不仅可以检查出是否有括号匹配错误，而且还知道具体错在哪个括号(多出的括号)，用行号和列号给出。

括号匹配的原则是把最近的左、右括号配成一对，所以括号匹配的常用方法是：遇到左括号入栈，遇到右括号出栈，出栈的左括号与当前的右括号是匹配的；若遇到右括号而

栈空(没有左括号)，则匹配错误(右括号多余)，若文件结束(不再可能有右括号)，栈非空，此时亦匹配错误(左括号多余)。为了给出错误括号位置，就需要在左括号入栈时同时将行号和列号入栈。

下面具体分析该算法。

首先，把栈置空，置 EOF 为 false，并从文件中读取第一字符到 ch。然后进入循环，循环体指向一次处理一个 ch。进入循环，利用 kind 函数算出 ch 的类型 k，接下来就是一大堆的填空了。填空比较集中，增加了一定解题难度。其大致结构如下：当 k 等于什么的时候把什么入栈，当 k 等于什么的时候且栈不为空的时候出栈，如栈为空打印错误信息，如果都不是则继续从文件中读取下一个字符再次进入循环。根据上面提到的括号匹配方法很容易得出：入栈应是在类型 k 为 1(ch 为左括号)时进行，出栈应是在类型 k 为 2(ch 为右括号)时进行。故空(1)处应填 1，空(4)处应填 2。那么，到底是将什么压入栈呢？根据上面说明，可以猜测是将行号和列号入栈，空(4)处之后的出栈并未用到栈的内容，循环结束后有语句“row←pop(); col←pop();”，根据 push 与 pop 的对应关系，考虑到栈的后进先出特性，可得空(2)处应填 col，空(3)处应填 row。注意：两个顺序不能颠倒。

完成算法 1 后，算法 2 的分析就简单多了。同理，根据栈的后进先出特性，空(5)处为 col，空(6)处为 row。另外，这里涉及的括号有 3 种，而左方括号只能与左方括号匹配，不能与左花括号匹配，也就是说入栈时还需将相应的括号类型入栈，这样才能在出栈时正确判断是否匹配。故空(7)处应填 k。

【问题 2】

判断条件 1 为真时进行入栈操作，因此判断条件 1 就是判断字符是否为左括号(包括左圆括号、左方括号、左花括号)，故选(b)。

判断条件 2 和判断条件 3 是联系在一起的，综合体现了括号匹配条件：栈中有与当前括号同类型的左括号。特别地，当栈空时，即匹配失败。根据题述判断条件 2 为假时无须对判断条件 3 进行判断。因此判断条件 2 为“栈不空”，选(e)；判断条件 3 为“栈顶元素表示的是与当前字符匹配的左括号”，选(f)。

【试题六】

答案：

(1) private。 (2) static。 (3) virtual。 (4) strategy*。 (5) strategy->nextHand()。

解析：

Hand 类要保证只产生 3 个实例，就要求不能随便生成 Hand 类，因此其构造方法需要是 private 型的，故空(1)处应填 private。

看空(3)处，由后面的“=0”易判知 nextHand()函数是纯虚函数，故空(3)处应填 virtual。

再来看空(2)处，由对 getHand()方法的调用方式 Hand::getHand(rand()%3)及 Hand::getHand(handvalue)，可知该方法是类 Hand 的静态方法，故空(2)处应填 static。

由语句 this->strategy = strategy 可知，this->strategy 与 strategy 是同数据类型，this 表示自身指针，而 this->strategy 是 Strategy*类型，故空(4)处应填 Strategy*，注意是指针。

方法 nextHand()是“向战略请示手势”，取得手势是通过其所采用的“策略”实现的，故空(5)处应填 strategy->nextHand()。注意指针写法。

【试题七】

答案：

(1) private。 (2) static。 (3) abstract。 (4) Strategy。(5) strategy.nextHand()。

解析：

Hand 类要保证只产生 3 个实例，就要求不能随便生成 Hand 类，因此其构造方法需要是 private 型的，故空(1)处应填 private。

看空(3)处，nextHand()是接口 Strategy 的方法，应为抽象方法，故空(3)处应填 abstract。

再来看空(2)处，由对 getHand()方法的调用方式 Hand.getHand(handvalue)及 Hand.getHand(random.nextInt(3))，可知该方法是类 Hand 的静态方法，故空(2)处应填 static。

由语句 this.strategy = strategy 可知，this.strategy 与 strategy 是同数据类型，this 表示自身引用，而 this.strategy 是 Strategy 类型，故空(4)处应填 Strategy。

方法 nextHand()是“向战略请示手势”，取得手势是通过其所采用的“策略”实现的，故空(5)处应填 strategy.nextHand()。

7.2.4 样卷四解析

软件设计师样卷四答案与解析

【试题一】

答案：

【问题 1】

0 层图中文件“座位表”是局部数据文件，不必画出。

【问题 2】

(1) 图 7-27 中从加工 1.1“检查订票单”到文件“列车时刻表”的数据流。

(2) 图 7-27 中从文件“旅客信息表”到加工 1.3“填写取票单”的数据流。

(3) 图 7-28 中从加工 2.1“检查取票单”到文件“旅客信息表”的数据流。

【问题 3】

不可以。从 0 层图中可以看出，加工 3“查询处理”仅有从文件“订票信息表”的输入数据流，而从说明中的文件组成可以看出，订票信息表仅记录了旅客订票和取票的信息，不能知道总的座位数，因此加工 3 能查询已订购和已售出的车票情况，而不能查询出剩余票的情况。

解析：

【问题 1】

“不必画出”是指在某层数据流图中，只画流程图中各加工之间的公共数据文件，隐藏某加工的局部数据文件。这个规则只是为了使整个数据流图的层次结构更科学、更清晰，然而画出“不必画出的数据文件”对数据流图不会造成理解错误。在 0 层图中有文件“列车时刻表”“订票信息表”“旅客信息表”和“座位表”，其中“座位表”是加工 1“顾客订票”的局部数据文件，所以不必画出。

【问题 2】

本问题是要找出错误的数据流。解决这种问题的关键是父图与子图的平衡，即子图的输入/输出数据流与父图相应的加工的输入/输出数据必须一致。

从 0 层图中可以看到对于加工 1“顾客订票”，有到文件“旅客信息表”的输出数据流，从文件“列车时刻表”得到的输入数据流，以及与文件“订票信息表”的输入/输出数据流。而加工 1 子图中只有从加工 1.1 到文件“列车时刻表”的数据流，与父图不一致，因此是错误的，应该为从文件“列车时刻表”到加工 1.1 的数据流。同理，从文件“旅客信息表”到加工 1.3 的数据流也与父图不一致，应该改为从加工 1.3 到文件“旅客信息表”的数据流。

0 层图中加工 2“顾客取票”中存在从文件“旅客信息表”到加工 2 的数据流，而加工 2 子图中从加工 2.1 到文件“旅客信息表”的数据流是与 0 层图相悖的，因此也是错误的。应该改为从文件“旅客信息表”到加工 2.1“检查信息表”的数据流。

【问题 3】

每个加工的功能要从提供给该加工的文件的信息量决定。从 0 层图中可以看出，加工 3“查询处理”仅有从文件“订票信息表”的输入数据流，而从说明中的文件组成可以看出，订票信息表仅记录了旅客订票和取票的信息，不能知道总的座位数，因此加工 3 能查询已订购和已售出的车票情况，而不能查询出剩余票的情况。要想查询出剩余票的情况，需要利用文件“座位表”来了解车次座位的总的情况，再利用文件“订票信息表”的已订和已售车票信息共同得出剩余票信息。

【试题二】

答案：

【问题 1】

修改后的数据模型如图 7-71 所示。

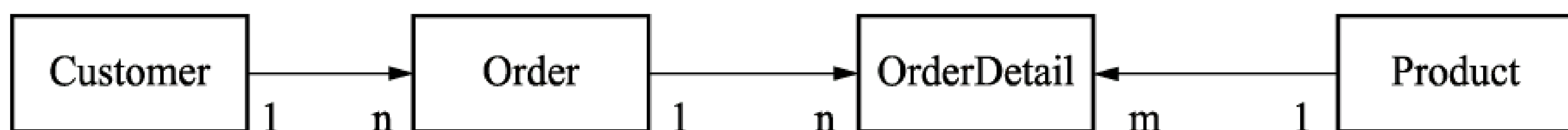


图 7-71　修改后的数据模型

【问题 2】

(1) 因为数据库中没有记录订货时产品的单价，也没有记录订货的总金额，所以一旦产品单价发生变化，那么计算用的单价就是变化后的单价。

(2) 在 OrderDetail 中增加一个数据项：订货时的单价(或者在 Order 中增加一个数据项：总金额)。

【问题 3】

OrderNo,ProductNo。

【问题 4】

(1) A 或 AS A。　　　(2) NOT EXIST。

解析：

本题考查了数据库系统中的数据模型、E-R 模型转换为关系模式、主键和 SQL 语句的有关知识。

【问题 1】

按照 E-R 模型转换为关系模式的规则，一个 1∶1 或 1∶n 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。而 m∶n 联系转换为一个独立的关系模式时，与该联系相连的各实体的键及关联本身的属性均转换为关系的属性，而关系的键为各实体键的组合。

从给出的数据模型中可以看到存在 Order 和 Product 之间的多对多关系，在现实数据库设计时，必然要加一个弱实体集。实体和属性之间一般遵循两条原则：一是“属性”必须是不可再分的数据项，不能具有需要描述的性质；二是“属性”不能与其他实体有联系。所以根据这两条，订单的情况不能作为订单 Order 的属性处理，而应该上升为实体。所以 Order 和 Product 之间的多对多关系应该拆分出一个名为 OrderDetail 的关系模式，从而形成 Order 与 OrderDetail 间的一对多关系，同时形成 OrderDetail 与 Product 间的多对一关系。

详见答案。

【问题 2】

(1) 首先从题目中找到甲公司的要求：“产品的单价发生变化时，应及时修改产品表中的单价数据。客户购货计价采用订货时的单价。订货后，即使单价发生变化，计算用的单价也不变。”而给出的几个关系中，只有 Product 中记录了产品单价 UnitPrice，一旦价格发生变化，用户当时订单的单价也会随之变化，所以无法实现甲公司的要求。

(2) 应该在 OrderDetail 中增加订货时的单价或者在 Order 中增加总金额这个数据项。这样，即使产品的单价发生了变化，客户仍然可以按照原来订货时的价格提货。

【问题 3】

本题中 Product 和 Order 是多对多的关系，Order 的键是 OrderNo，Product 的键是 ProductNo，所以拆分后形成的 OrderDetail 的数据模型为 OrderDetail(OrderNo，ProductNo，Quantity)，其中关键项为 OrderNo 和 ProductNo。

【问题 4】

SQL 中用以下两种格式为表或视图指定别名：“表名 as 别名”或者“表名 别名”。由题目中的 SQL 语句可以看出 OrderDetail 的别名为 B，Order 的别名为 C；根据“C.CustomerNo = A.CustomerNo”和题目中给出的几个关系可知，有 CustomerNo 项的关系仅有 Customer 和 Order，“C”是 Order 的别名，则“A”肯定为 Customer 的别名。所以空(1)处填写 A 或 AS A。

题意要求“查询没有订购产品代码为‘1K10’的产品的所有客户名”，而括号中 SQL 语句表示订购产品代码为“1K10”的产品的所有客户名，显然空(2)处应填 NOT EXIST。遇到此类填写约束条件的，一定要结合上下文考虑。

【试题三】

答案：

【问题 1】

电影代码、电影片名、导演。

【问题 2】

(A)订购音像。　(B)联系客户。

【问题 3】

补充后的类图如图 7-72 所示。

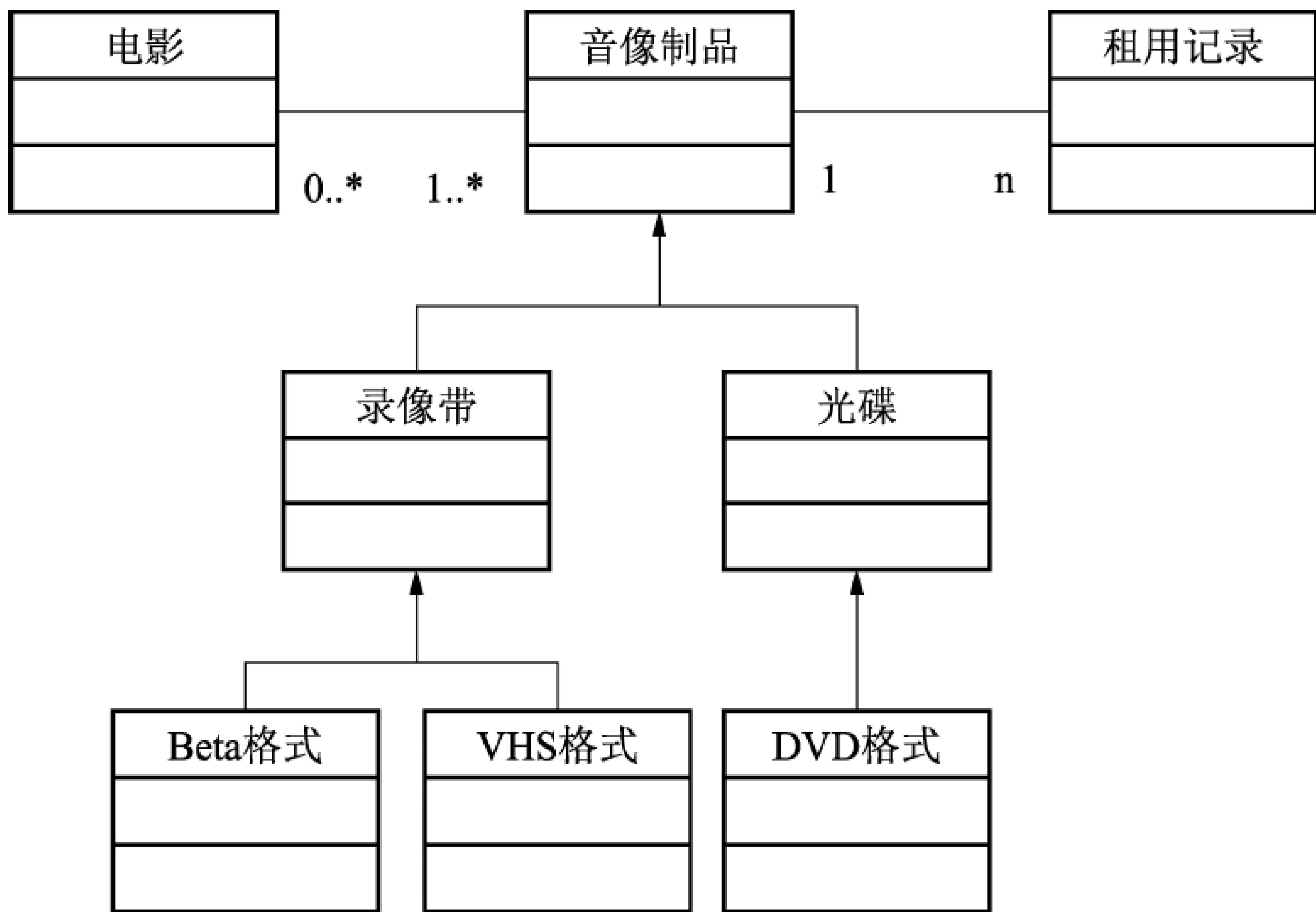


图 7-72　补充后的类图

解析：

【问题 1】

根据“由于同一个电影片名可能由于不同的导演而有不同的版本，因此电影用电影代码区分，而不用电影片名；同一个版本有多份副本，因此音像制品用一个唯一的编号标识”，可得电影的主要属性有：电影代码、电影片名、导演。注意：“编号”不是“电影”的属性，而是“音像制品”的属性。

【问题 2】

图 7-31 是该系统的用例图。根据题述，“音像商店的店员负责订购音像、联系客户、音像上架，并对客户的询问给出答复”，易知缺失的用例为“订购音像”和“联系客户”。

【问题 3】

UML 中的关系有依赖、关联、泛化和实现。依赖(Dependency)是两个事物之间的语义关系，其中一个事物发生变化会影响另一个事物的语义。关联是一种结构关系，聚集(Aggregation)是一种特殊类型的关联，描述了整体和部分之间的结构关系。泛化(Generalization)是一种特殊/一般关系。实现(Realization)是类元之间的语义关系，其中一个类元制定了由另一个类元保证执行的契约。

“音像制品”是电影的载体，自然与“电影”有关联。关联度是多少呢？先来看“音像制品”与“录像带”及“光碟”间的关系，“录像带”及“光碟”都是“音像制品”的不同存储格式，因此“录像带”及“光碟”都是“音像制品”的特殊化。再回到“音像制品”与“电影”的关联度，“录像带”只存储一个电影版本，而“光碟”可以存储多个版本，因此一个“音像制品”有一个或多个“电影”，一个“电影”可以存储于多个“音像制品”中(当然也可能没有)。

一个“音像制品”对应多个“租用记录”，一个租用记录只对应一个“音像制品”。

【试题四】

答案：

(1) knap(s−w[n], n−1)。

(2) knap(s, n−1)。

(3) top>=1 && !k 或 top>0 && k == 0。

(4) x.s − w[x.n−−]。

(5) stack[++top]。

(6) rep = 0。

解析：

“背包问题”是历年试题考得最多的一个经典问题，可由递归和非递归两种算法实现。不管是递归还是非递归，程序算法的思路都是依次考查每个物品。对物品 i，考查两种可能情况。首先，考查物品被选择的情况，这种可能性当且仅当包含它不会超过方案总重量的限制时才是可行的；物品 i 被选择后，继续考查下一个物品。其次，还要考查物品 i 不被选择的情况，这种可能性当且仅当不包含物品 i 时，也有可能找到价值更大的方案；考查完物品 i 后，也要继续考查下一个物品。

程序 1 采用递归算法实现“背包问题”。对每个物品 i，考查选择放入和不放入背包两种情况。函数 knap(int s, int n)中，形参 s 是考查完物品 i 后背包还能装载的重量，n 是考查完物品 i 后下一个待考查的物品。每次选择一个物品放入背包，那么剩余的物品和背包剩余重量又构成一个“背包问题”。根据注释，空(1)处是考查物品 n 放入背包的情况，既然放入背包，则背包剩余可装重量为 s−w[n]，继续考查物品 n−1。这点可从主函数的调用形式 knap(S, N)分析出。故空(1)处应填 knap(s−w[n], n−1)。空(2)处是考查物品 n 不放入背包的情况，既然不放入背包，则背包可装重量仍为 s，继续考查物品 n−1。故空(2)处应填 knap(s, n−1)。

程序 2 采用非递归算法实现“背包问题”。程序使用栈(即数组 stack 表示)来保存已经考查过的物品(函数的递归调用其实也是通过栈实现的)。经分析，结构变量 KNAPTP 表示经过考查的物品：分量 s 表示考查过该物品后，背包所能盛放的物品的重量；分量 n 表示待考查的下一个物品在数组 w 中的下标；分量 job 表示物品当前的状态，job 等于 1 表示物品 n 可以放入背包，job 等于 2 表示物品不能放入背包，在以后的选取中将不再考虑该物品，初始时 job 等于 0 表示背包中没有放入任何物品。据注释“k=1 时则求得一组解”可知 k 为是否求得解的标志：k=0 表示没有解，继续求解。rep 是一个标志变量，等于 0 表示结束当前的动作，等于 1 表示继续进行当前的动作；当栈顶物品不能装入背包时，将 rep 置为 0，表示下一步不再从数组 w 中取物品。rep 初值为 1。x 为工作节点。

while((3))循环体内的语句可以肯定是考查各个物品 n 的选择情况。对物品 n，先考查将物品放入背包的情况。显然如果物品 n 满足放入背包的条件，则空(4)处和空(5)处完成将物品放入背包的操作，其中空(4)处应该是要将工作节点 x 的分量 s 值减去所考查物品的重量，且 n 要减 1，修改背包可容纳物品的重量和设置下一个待考查物品，而空(5)处则需要将修改后的工作节点 x 送到栈顶，将下一个待考查的物品入栈。故空(4)处应填 x.s – w[x.n– –]，空(5)处应填 stack[++top]。

if(!k)后的程序段是处理所考查的物品不满足放入背包的条件时的情况(rep=0，while(!k && rep)循环结束)，则将该物品从背包中取出，修改其 job 值为 2，用以标记该物品不能放

入背包。修改完后跳出 while(top >= 1 && rep)循环，因此需要将 rep 置为 0，用以结束循环。故空(6)处应填 rep = 0。

【试题五】

答案：

(1) (*a!='\0')&&(*b!='\0')。　(2) a++。　(3) b++。

(4) *w=='\0'。　(5) *(++w)=t。　(6) a++。　(7) *(++w)='\0'。

(8) s[i]>s[j]。　(9) s[j]=t。　(10) strmerge(s1,s2,s3)。

解析：

根据题意，对字符串的处理分为 4 步：第一步是从键盘上输入两个字符串；第二步是将两个字符串分别排序；第三步是将字符串合并；第四步是显示处理结果。

第一步和第四步好办，关键是第二步和第三步的处理，下面分别加以说明。

字符串排序是指将一个字符串中各个字符按照 ASCII 码值的大小排序。例如，字符串“Beijing”由小到大的排序结果应该是"Begiijn"。排序算法很多，这里介绍快速排序算法。这里，使用简单的冒泡排序算法，即将字符串中的每一个字符一个个进行比较，找出最小的字符，然后再在剩下的字符中找最小的字符，……。例如，字符“Beijing”的排序过程如下。

第一次将字符“Beijing”中的每一个字符'B'、'e'、'i'、'j'、'i'、'n'、'g'进行比较，找到最小的字符'B'。

第二次在剩下的字符'e'、'i'、'j'、'i'、'n'、'g'中，找到最小的字符'e'。

第三次在剩下的字符'i'、'j'、'i'、'n'、'g'中，找到最小的字符'g'。

……

第三步是合并字符串，合并后的字符串仍然由小到大排序。由于待合并的两个字符串已经排好序，假定两个排好序的字符串分别为 A 和 B，合并后的字符串为 C，要使合并后的字符串仍然由小到大排序，可采取下述步骤。

(1) 从前往后取 A 中的字符，并按从前往后的顺序与 B 中的字符比较，若 A 中的字符较小且与 C 中当前字符不等，则将该字符存入 C，并移到 A 的下一个字符，继续与 B 中的字符比较。

(2) 若 A 中的字符较大且 B 中的字符与 C 中当前字符不等，则将 B 中的字符存入 C，并移到 B 的下一个字符，继续与 A 中的字符比较。

(3) 若 A 与 B 中的字符相等且与 C 中当前字符不等，则将 A 或 B 中的字符存入 C，并将 A 和 B 均移到下一个字符。

(4) 若 A 或 B 字符串到达末尾，则将 B 或 A 的剩余字符逐个加到字符串 C 中。添加每个字符前都要将添加的字符与 C 中当前字符进行比较，不等时才加到 C 中。

需要注意的是，A、B 和 C 3 个字符串均可以用字符数组来表示，C 数组的长度不能小于 A、B 两数组的长度之和。另外，判别字符串是否结尾的方法是：判断从 A 或 B 中取出的字符是否为'\0'，所有字符串都是以'\0'结尾的。

【试题六】

答案：

(1) virtual。 (2) DP2::drawline(x1, x2, y1, y2)。 (3) Drawing。

(4) _dp->drawLine(x1, y1, x2, y2)。 (5) Shape(dp)。

解析：

由函数 drawLine()结尾的 “=0” 易知，空(1)处应填 virtual。

空(2)处是调用 DP2 系统的相应方法，可参照 DP1 的对应函数的函数体，但要注意参数不完全相同，应填 DP2::drawline(x1, x2, y1, y2)。

_dp 属性是用来存储 Drawing 对象的，参照 Shape 的构造函数可确认这一点，空(3)处应填 Drawing*。

Shape 类的 drawLine 方法是通过调用 Drawing 对应的方法来实现所需要的功能，因此空(4)处应填_dp->drawLine(x1, y1, x2, y2)。

空(5)处显然是基类构造函数，应填 Shape(dp)。

【试题七】

答案：

(1) abstract。 (2) DP2.drawline(x1, x2, y1, y2)。 (3) Drawing。

(4) _dp.drawLine(x1, y1, x2, y2)。 (5) super(dp)。

解析：

由于类 Drawing 的 drawLine()方法是 abstract 的，因此 Drawing 要么是接口，要么是抽象类，在此为抽象类，故空(1)处应填 abstract。

空(2)处是调用 DP2 系统的相应方法，可参照 DP1 的对应函数的函数体，但要注意参数不完全相同，应填 DP2.drawline(x1, x2, y1, y2)。

_dp 属性是用来存储 Drawing 对象的，参照 Shape 的构造函数可确认这一点，空(3)处应填 Drawing。

Shape 类的 drawLine 方法是通过调用 Drawing 对应的方法来实现所需要的功能，因此空(4)处应填_dp.drawLine(x1, y1, x2, y2)。

空(5)处显然是基类构造函数，应填 super(dp)。

7.2.5 样卷五解析

软件设计师样卷五答案与解析

【试题一】

答案：

【问题 1】

0 层图中的“房租文件”和“交费文件”是局部文件，可不必画出。

【问题 2】

加工 1 子图中，遗漏了从“住户基本信息文件”到加工 1.1 的数据流。

加工 1 子图中，加工 1.6 遗漏了输出数据流“住房分配表”。

加工 2 子图中，加工 2.1 遗漏了输入数据流“月附加费表”。

加工 2 子图中，加工 2.4 遗漏了输出数据流“收据”。

【问题 3】

交费凭证中有非法字符。

交费文件中不存在与之对应的交费凭证。

解析：

房产管理系统是描述住户与物业管理委员会之间的数据输入与输出的变换过程。试题明确指出了顶层图的正确性，所以根据试题说明与顶层图确定系统的基本功能如下。

(1) 分类处理用户的入住单，更新住户基本信息，结算分户或换房前的房租，以及制作住房分配表。

(2) 计算住户月租费，发出交费通知单，处理住户交费，以及制作住房分配表和交费情况表。

在 0 层图中有“住户基本信息文件”“房租文件”和“交费文件”3 个文件。其中“房租文件”“交费文件”为加工 2 的局部数据文件，根据原则，这两个文件在 0 层图中不必画出，但在问题 2 中给出了可能有遗漏数据流的情况，还须分析加工 1 子图，以确定该加工没有遗漏使用这些文件的数据流。下面就通过分析加工 1 子图的处理流程，确定遗漏的数据流及上面关于局部文件的假设。

加工 1 子图由 6 个加工组成，即入住单校验、按入住性质分类、入住、分户处理、换房处理，以及制作住房分配报告。

加工 1.1 对入住单进行校验，反馈不合法的入住单。那么加工 1.1 根据什么来校验入住单？该加工没有提供任何校验数据，如果不借助外部的数据信息，只能检查入住单数据中的一般性错误，如录入的数据是否含有非法字符、数据格式是否满足系统的约定等；另外还需检查录入的入住单数据的一般性错误，保证录入的入住单数据是合法的，根据一般的知识经验，对某个输入合法性的校验，需要借助某些外部数据文件，因此可以判断该加工遗漏了到某数据文件的输入数据流。

对于合法的入住单，加工 1.2 对它进行分类处理，分成 3 类：新住户(加工 1.3)、分户(加工 1.4)、换房(加工 1.5)，每个处理更新住户基本信息文件。那么这里可以确定 1.1 加工的外部数据应该是“住户基本信息文件”，也就是加工 1.1 根据“住户基本信息文件”进行合法性检查。

加工 1.3 处理新住户，只要把住户信息写入文件即可。而加工 1.3 与加工 1.4 除了更新住户基本信息文件外，还应立即对这些住户做月租费计算，以结算分户或换房前的房租。

这里加工 1.4 和加工 1.5 可能直接依据“房租文件”和“交费文件”计算房租，但这两个加工的输出数据流是两个输出通知，也就是房租的计算交给加工“收费管理”来完成，这样简化了处理流程和系统结构。所以可以确定这两个文件不是加工 1 的数据文件，因此没有必要在 0 层图中画出。

再来看整个处理流程的输入与输出数据，发现整个流程与加工 1 有输出不平衡。少了“住房分配表”的输出，根据加工的命名可以判断是由加工 1.6 制作住房分配表输出。加工 2 子图由计算月租费、调整房租、交费凭证验证、制作数据及制作交费报告 5 个加工组成，

也即整个流程完成房租计算(加工 2.1 与加工 2.2)、交费处理(加工 2.3 与 2.4)、报表制作(加工 2.5)3 个功能。

加工 2.2 对房租调整表进行处理进而生成房租文件，作为加工 2.1 的输入数据；加工 2.1 进行房租的计算，生成交费文件及产生交费通知单。那么房租计算需要哪些数据呢？试题说明给出“根据物业管理委员会提供的月附加费表和房租调整表，计算每家住户的月租费”的条件，房租调整表由加工 2.2 加工生成房租文件，再由加工 2.1 读入，所以加工 2.2 不存在遗漏数据流；而月附加费表只有直接输入，所以加工 2.1 遗漏了“月附加费表”输入数据流。住户持交费凭证交费时，由加工 2.3 根据“交费文件”进行凭证的合法性检查。这就需要检查交费凭证是否在文件中存在，交费数据与文件是否一致等数据个别性错误，同时还要判断输入的凭证数据是否含有一些非法字符等一般性错误。经检查后产生“合格交费凭证”数据流给加工 2.4 制作数据并输出收据给住户，显然加工 2.4 遗漏了“收据”输出数据流。加工 2.5 定期根据“交费文件”制作交费报告提供给物业管理委员会，不存在遗漏数据流的问题。再分析整个细化流程图的输入与输出数据流，确定该流程图不存在父图和子图输入和输出数据的不平衡。

【试题二】

答案：

【问题 1】

“借阅”联系是“多对多”联系，外键有借书证号、图书流水号。

“借阅”关系模式不存在主键，因为同一本书读者可以重复借阅，即使加上“借书日期”亦不能唯一标识，读者可以在同一天借阅并归还再借阅，应增加一个属性(如借阅流水号)来标识。

【问题 2】

图书(分类目录号，书名，作者，内容摘要，价格，购书日期)，主键：分类目录号。

副本(图书流水号，分类目录号)，主键：图书流水号。

【问题 3】

(1) IN。 (2) Borrow。 (3) 归还标记= "false"。 (4) LIKE。

解析：

【问题 1】

根据“一个读者最多可以同时借阅 5 本图书……一本书可能供多位读者借阅，同一本书读者可以重复借阅”可知，“借阅”联系是“多对多”联系。

要特别分清“一本图书”与“一种图书”。一本图书是由流水号标识的，一种图书是由分类目录号标识的。对“借阅”联系，借阅的是具体的某本书，因此外键有借书证号、图书流水号。

“借阅”关系模式不存在主键，因为同一本书读者可以重复借阅，即使加上“借书日期”亦不能唯一标识，读者可以在同一天借阅并归还再借阅，应增加一个属性(如借阅流水号)来标识。

【问题 2】

由于同一个分类目录号(同一种图书)有多个副本，同一个分类目录号具有相同的很多信息，如书名、作者、内容摘要、价格等，同一种书中的不同副本由图书流水号区分，故可

分解如下。

图书(分类目录号，书名，作者，内容摘要，价格，购书日期)，主键：分类目录号。

副本(图书流水号，分类目录号)，主键：图书流水号。

【问题 3】

空(1)处是引出子查询的，该类连接词有 IN、NOT IN、EXISTS、NOT EXISTS。EXISTS 引出的子查询一般是 SELECT *型，故排除；再据语意分析应填 IN。

子查询的语义是“查询当前所借阅的图书流水号(即尚未归还的图书)”，因此应从 Borrow 表中查询，而且归还标记应为 false，故空(2)处应填 Borrow，空(3)处应填“归还标记 = "false"”。

对字符串进行的操作通常是使用操作符 LIKE 的模式匹配，正符合题意，故空(4)处应填 LIKE。

【试题三】

答案：

【问题 1】

属性：姓名、性别、身份证、联系电话。

方法：预订、入住、结账。

【问题 2】

(1) 0..1。(2) 1..*。(3) 1。　(4) 0..*。

【问题 3】

A：空闲。　　B：占用。　　C：入住。　　D：取消预订。

解析：

【问题 1】

“客人”类是“散客”类和“团体”类的泛化，具有二者的公共属性和公共方法。比对二者属性及方法得，“客人”类属性有姓名、性别、身份证、联系电话，方法有预订、入住、结账。

【问题 2】

散客入住时只改变一个客房状态，而团体入住时则有可能改变多个客房状态；客房状态改变不一定是住宿导致的，客房维修同样改变客房状态。因此空(1)处应填 0..1，空(2)处应填 1..*。

客人可以有多项服务，但只需用一张服务列表，当然也可能不需要任何服务；而一张服务列表必然属于而且只属于一个住宿。因此空(3)处应填 1，空(4)处应填 0..*。

【问题 3】

“维修”完成后客房处于“空闲”状态，故状态 A 为“空闲”；客人入住后，客房由“空闲”转为“占用”，故状态 B 为“占用”；状态“已预订”经“入住”转为“占用”，故状态 C 为“入住”；状态“已预订”经“取消”转为“空闲”，故状态 D 为“取消预订”。

【试题四】

答案：

【问题 1】

此程序是一个排序程序。它将数组 a 中的元素进行从小到大排序。

【问题 2】

输出的其实就是排序的前 3 趟中间结果。

第 1 趟：8,2,10,7,15,4,13。

第 2 趟：2,7,8,4,10,13,15。

第 3 趟：2,4,7,8,10,13,15。

解析：

此排序方法称为奇偶交换排序。

排序过程为：第 1 趟对所有的奇数 i，将 a[i]与 a[i+1]进行比较，若 a[i] > a[i+1]，则将两者交换；第 2 趟对所有偶数 i，将 a[i]与 a[i+1]进行比较，若 a[i] > a[i+1]，则将两者交换；以后重复上述两趟过程，以此类推直到整个序列有序为止。只要看懂了程序流程图，两个问题都比较容易回答。

【试题五】

答案：

(1) j++。　(2) j*=2 或 j=k*2。　(3) j=n+1 或 break。

(4) adjust(i,n)。　(5) adjust(1,k−1)。

解析：

函数 adjust(i,n)是把以 R[i]($1 \le i \le \lfloor i/2 \rfloor$)为根的二叉树调整成堆的函数，假定 R[i]的左、右子树已经是堆，程序中的 r 是在主函数中说明的结构数组，它含有要排序的 n 个记录。

```
void adjust(i,n)
int i,n;
{
int k,j;
element extr;
extr=r[i];
k=i;
j=2*i;
while(j<=n)
{if((j<n)&&(r[j].key<r[j+1].key))
j++;
if(extr.key<r[j].key)
{
r[k]=r[j];
k=j;
j*=2;
}
else
j=n+1;
}
r[k]=extr;
}
```

让 i 从$\lfloor i/2 \rfloor$逐步减到 1，反复调用函数 adjust，便完成建立初始堆的过程。堆排序程序 heapsort 如下：

```
void heapsort(r,n)
list r;
int n;
{
int i,l;
element extr;
for (i=n/2;i>=1;--i)
adjust(i,n); /*建立初始堆*/
for(k=n;k>=2;k--)
{
extr=r[1];
r[1]=r[k];
r[k]=extr;
adjust(1,k-1);
}
}
```

函数 heapsort 的第一个 for 循环建立初始化。设待排序的 n 个记录组成一棵深度为 h 的完全二叉树，因此有 $2h-1 < n \leqslant 2h+1-1$，即 $2h \leqslant n < 2h+1$。完全二叉树的第 i 层(设根节点的层次为 0)上最多有 2i 个节点，对每个非叶节点，都要调用过程 adjust，同时可能移动节点(向下移动)，第 i 层上的节点可能要移动的最大距离为 h−i，若设向下移动一层花费的时间为 c，则第 i 层 2i 个节点调整的时间最多为 c*(h−i)* 2i。

对第二个 for 循环，调用 adjust 函数 n−1 次，每次总是由根向下调整，调整的最大距离为 $\log_2 n$(实际上，调整的距离是逐渐变小的)，所以总的时间不多于 $c*(n-1)\log_2 n=O(\log_2 n)$。堆排序总的时间为 $O(n)+O(n\log_2 n)=O(\max(n,n\log_2 n))=O(n\log_2 n)$。

算法需要的存储空间是存储 n 个记录的数组以及少量暂存单元。

堆排序算法是不稳定的。

【试题六】

答案：

(1) public Figure。　　(2) height * width。　　(3) public Rectangle。

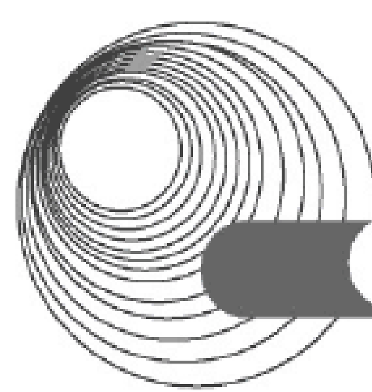
(4) height = this->width = width。　　(5) public Figure。

解析：

根据题述“抽象类 Figure 提供了一个纯虚函数 getArea()，作为计算上述 3 种图形面积的通用接口”，可知类 Rectangle 应继承自类 Figure，并实现其抽象方法 getArea。故空(1)处应填 public Figure。

getArea 方法是计算面积的，空(2)处要返回该类代表的图形的面积。类 Rectangle 代表矩形，而矩形面积的计算公式是长与宽的乘积，类 Rectangle 的成员变量 height 和 width 分别代表了长和宽，故空(2)处应填 height*width。

类 Square 按理应该继承自 Figure，但并未实现其抽象方法 getArea，若继承自 Figure 则不能被实例化，而题中 main 函数中已将其实例化，不符合题意，因此不能继承自 Figure。考虑到正方形其实就是长和宽相等的矩形，因此 Square 可从 Rectangle 继承，故空(3)处应填 public Rectangle。这样，计算面积仍用 Rectangle 的 getArea 方法，这就要求将 height 和



width 正确赋值，题中已将这两个成员变量声明为 protected，因此这两个变量继承为子类的变量，故空(4)处应填 height = this->width = width。

空(5)处同空(1)处，应填 public Figure。

【试题七】

答案：

(1) Point center。 (2) new Point(xValue, yValue)。 (3) this(xValue, yValue, r)。

(4) extends Ball。 (5) super(xValue, yValue, r, c)。

解析：

在 Ball 类的有参数构造函数中，对成员变量 center 通过调用 Point 类的构造方法初始化，而 center 在 Ball 类中尚未声明，结合注释可得空(1)处是将 center 变量声明为 Point 对象引用，故空(1)处应填 Point center，空(2)处是调用 Point 类的构造函数，根据题意，此处应将 xValue 和 yValue 作为参数调用 Point 类的有参数构造函数，故空(2)处应填 new Point(xValue, yValue)。根据注释，空(3)处是调用 Ball 类的有 3 个参数的构造方法，而其所在方法本身就是 Ball 类的一个构造方法，因此可用 this 来调用自身的构造方法，故空(3)处应填 this(xValue, yValue, r)。

根据题述“在 MovingBall 类的 toString 方法中，super.toString 调用父类 Ball 的 toString 方法输出 Ball 类中声明的属性值”，可知 MovingBall 类是 Ball 类的子类，因此空(4)处应填 extends Ball。

根据注释，空(5)处是调用父类 Ball 中具有 4 个参数的构造方法，通过 super 关键字实现，故空(5)处应填 super(xValue, yValue, r, c)。

7.2.6 样卷六解析

软件设计师样卷六答案与解析

【试题一】

答案：

【问题 1】

名称：当前日期。起点：系统时钟。

【问题 2】

(b) 读者文件。 (c) 借书文件。

【问题 3】

(d) 分类目录号。 (e) 图书流水号。 (f) {图书流水号}。

解析：

【问题 1】

加工 2 的输入数据流有“当前日期”和“有效的图书管理要求”。根据平衡原则，加工 2.1 的输入数据流(a)应为“当前日期”，其起点自然是“系统时钟”。

【问题 2】

加工 3.2“读者查询”需要“借书文件”及“读者文件”，而加工 3.3“图书查询”与

“读者文件”不相关，因此(b)处应填“读者文件”，(c)处应填“借书文件”。

【问题 3】

根据说明“填写借书单，包括读者号、欲借图书分类目录号”可得，借书单应包括“读者号”和“图书分类号”。故(d)处应填“分类目录号”。

还书时是根据“图书流水号”的，因此还书单应包括“图书流水号”。故(e)处应填“图书流水号”。

“目录文件”存储图书的情况，包括分类目录号、图书流水号、书名、作者、内容摘要、价格，可见“目录文件”需要存储图书流水号，而且“一种”书有多个副本，因此(f)处应填“{图书流水号}”。注意：大括号{}表示多个数据项。

【试题二】

答案：

【问题 1】

Athlete(ANo, AName, ASex, Age, ATeam)，主键为 ANo。

Iteam(INo, IName,ITime, IPlace)，主键为 INo。

Games(ANo, INo, Score, Credit)，主键为(ANo, INo)。

【问题 2】

(1) PRIMARY KEY ANo。

【问题 3】

(2) SUM(Credit)。 (3) IN。 (4) Athlete。

解析：

【问题 1】

E-R 模型向关系模式的转换应遵循以下原则。

(1) 每个实体类型转换成一个关系模式。

(2) 一个 1∶1 的联系(一对一联系)可转换为一个关系模式，或与任意一端的关系模式合并。若独立转换为一个关系模式，那么两端关系的码及其联系的属性为该关系的属性；若与一端合并，那么将另一端的码及联系的属性合并到该端。

(3) 一个 1∶n 的联系(一对多联系)可转换为一个关系模式，或与 n 端的关系模式合并。若独立转换为一个关系模式，那么两端关系的码及其联系的属性为该关系的属性，而 n 端的码为关系的码。

(4) 一个 n∶m 的联系(多对多联系)可转换为一个关系模式，两端关系的码及其联系的属性为该关系的属性，而关系的码为两端实体的码的组合。

(5) 3 个或 3 个以上多对多的联系可转换为一个关系模式，诸关系的码及联系的属性为关系的属性，而关系的码为各实体的码的组合。

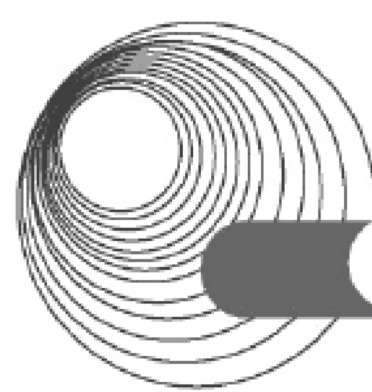
(6) 具有相同码的关系可以合并。

根据上述规则，可得以下关系模式：

Athlete(ANo, AName, ASex, Age, ATeam)，主键为 ANo。

Iteam(INo, IName,ITime, IPlace)，主键为 INo。

Games(ANo, INo, Score, Credit)，主键为(ANo, INo)。



【问题 2】

Athlete 表中 ANo 是主键，创建表时需要说明主键，故空(1)处应填 PRIMARY KEY ANo。

【问题 3】

创建表 Athlete 的 SQL 语句如下：

```
SELECT [ALL|DISTINCT]<目标列表达式>[,<目标列表达式>]…
        FROM <表名或视图名>[,<表名或视图名>]
        [WHERE <条件表达式>]
        [GROUP BY <列名 1>[HAVING<条件表达式>]]
        [ORDER BY <列名 2>[ASC|DESC]…]
```

子句顺序为 SELECT、FROM、WHERE、GROUP BY、HAVING、ORDER BY，但 SELECT 和 FROM 是必需的，HAVING 子句只能与 GROUP BY 搭配起来使用。SELECT 子句对应的是关系代数中的投影运算，用来列出查询结果中的属性，其输出可以是列名、表达式、集函数(AVG、COUNT、MAX、MIN、SUM)，DISTINCT 选项可以保证查询的结果集中不存在重复元组；FROM 子句对应的是关系代数中的笛卡儿积，它列出的是表达式求值过程中须扫描的关系；WHERE 子句对应的是关系代数中的选择谓词。

根据题意，空(2)处应填 SUM(Credit)，空(3)处应填 IN，空(4)处应填 Athlete。

【试题三】

答案：

【问题 1】

“自动售票机”类的主要属性有：编号、所在地铁站名。

【问题 2】

“地铁票”类的主要属性有：流水号、起始站、目的站、票价。

【问题 3】

状态 1：“目的站确认/票数输入”。　　状态 2：“票数确认/付款”。

状态 3：“出票/找零”。　　状态 4：“空闲”。

解析：

【问题 1】

根据“每一台售票机有一唯一编号”可知，自动售票机应有属性“编号”。“自动售票机只发售从该站起始的各种地铁票”，亦即自动售票机只发售其所在站为起始站的地铁票，故应有属性“地铁站名”。

【问题 2】

地铁票应有属性：起始站、目的站、票价、流水号。

【问题 3】

根据售票机状态变化的描述，易于得出答案。状态 1 应为“目的站确认/票数输入”，状态 2 应为“票数确认/付款”，状态 3 应为“出票/找零”，状态 4 应为“空闲”。

【试题四】

答案：

(1) (NODE*)malloc(sizeof(NODE))。 (2) makeTree()。 (3) *str == ','。

(4) putchar(t->val)。　　(5) walkTree(t->subTree[i])。

解析：

M 叉树本质上与二叉树没什么区别，只是二叉树最多只有两个子节点，用两个节点指针即可，而 M 叉树可以有 M 个子节点，需要用一个节点指针表(在此为一个数组)。树的递归性质亦没有改变，因此生成 M 叉树及遍历输出时，均可采用递归方式。

主程序读入列表，调用 makeTree()函数生成 M 叉树，再调用 walkTree()函数遍历输出。

子程序 makeTree()的功能是根据字符串 str 生成 M 叉树，列表的结构为：根节点的值+(子树 1，子树 2)。s 是一个 NODE*类型，在使用之前自然需要申请空间，故空(1)处应填(NODE*)malloc(sizeof(NODE))。然后，将 str 指针对应的字符赋给 s(根节点)，并将 str 指针加 1 指向下一个字符。接下来将子节点指针全初始化为 NULL。如果紧接着的字符是“(”，说明有子树，则将字符指针 str 加 1，递归生成第 k 棵子树；如果紧接着的字符是“,”(各子列表分隔符)，说明第 k 棵子树还有兄弟，继续生成第 k+1 棵子树；如果紧接着的字符是“)”，说明所有子树已生成完毕，结束循环。注意：子树也是 M 叉树。故空(2)处应填 makeTree()，空(3)处应填 str == ','。最后将 s 指针返回。

子程序 walkTree()的功能是变量 M 叉树输出列表。输出列表结构依然是：根节点的值+(子树 1，子树 2)。故空(4)处应该输出根节点 t 的值，应填 putchar(t->val)。接着判断是否有子树，若没有则返回；否则，递归输出各子树，用 for 循环实现，子树间用“,”分隔，所有子树输出完后，输出“)”。故空(5)处要递归输出第 i 棵子树，应填 walkTree(t->subTree[i])。注意：实参是子根节点 t->subTree[i]。

【试题五】

答案：

(1) pGraph->n−1。　　(2) mst[j].weight。　　(3) mst[i].StopVex。

(4) vy < vx。　　(5) pGraph->arcs[k]。

解析：

由注释“共 n−1 条边”可得，空(1)处应为 pGraph->n−1。

空(2)处相关程序段是选出权值最小的边，minWeight 表示的是最小权值，因此空(2)处应填 mst[j].weight。

由“vx 为刚加入最小生成树的顶点下标”可知，空(3)处应填 mst[i].StopVex。

邻接矩阵是压缩存储的，只存储上三角阵，因此下标需要进行转换。比较 if 及 else 块，可发现两算式区别在于 vx、vy 互换，由邻接矩阵的对称性可得空(4)处应填 vy < vx。

空(5)处相关程序段是进行调整，应填 pGraph->arcs[k]。

【试题六】

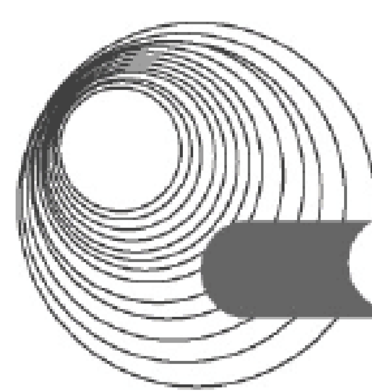
答案：

(1) virtual。　　(2) new XXCircle。　　(3) displayIt()。

(4) Shape*。　　(5) getShapeInstance(type)。

解析：

由“=0”可轻易判知 display()函数是一个纯虚函数，因此空(1)处应填 virtual。



由题设，Circle 实例化时须先实例化一个 XXCircle 对象，而 pxc 正好也是 XXCircle 对象指针，故空(2)处应填 new XXCircle。

Circle 在 Circle::display()中充当适配器的角色，它所做的就是将消息转发给 XXCircle 实例，display()是“显示”消息，故调用 XXCircle 的相应方法，空(3)处应填 displayIt()。

方法 getShapeInstance(int type)的返回值有 new Line、new Square 及 new Circle，参照类的层次结构，可得空(4)处应填 Shape*。注意是指针。

Factory 类仅定义了一个方法 getShapeInstance，而此处语义正是取得一个形状进行显示，故空(5)处应填 getShapeInstance(type)。

【试题七】

答案：

(1) abstract。　(2) implements。　(3) new XXCircle()。

(4) Shape。　(5) getShapeInstance(type)。

解析：

Shape 是接口，其中的方法都是抽象方法，故空(1)处应填 abstract。

Shape 是接口，故空(2)处应填 implements，表示实现某个接口。

初始化一个 XXCircle 实例，空(3)处应填 new XXCircle()。

方法 getShapeInstance(int type)的返回值有 new Line()、new Square()及 new Circle()，参照类的层次结构，可得空(4)处应填 Shape。

Factory 类仅定义了一个方法 getShapeInstance，而此处语义正是取得一个形状进行显示，故空(5)处应填 getShapeInstance(type)。

7.2.7 样卷七解析

软件设计师样卷七答案与解析

【试题一】

答案：

【问题 1】

数据流名称：成绩清单。起点：阅卷站。终点：统计成绩。

【问题 2】

数据流名称：报名单。起点：考生。终点：检查报名单。

【问题 3】

数据流名称：合格标准。起点：考试中心。终点：审定合格者。

错误数据流的起点：制作通知单。终点：考生名册。

解析：

【问题 1】

对于分层数据流图，一定要注意平衡原则，即父图与子图数据流一致。

仔细与顶层数据流图比对，可发现 0 层数据流图中缺失了数据流“成绩清单”，其起

点应为“阅卷站”，终点为加工 2“统计成绩”。

【问题 2】

同问题 1。可得加工 1 子图(图 7-55(a))中缺失了数据流“报名单”，其起点自然是“考生”，终点应为加工 1.1“检查报名单”。

【问题 3】

同问题 1。可得加工 2 子图(图 7-55(b))中缺失了数据流“合格标准”，其起点为“考试中心”，终点为加工 2.2“审定合格者”。

错误数据流多半是表现在文件读出/写入上。加工 2.3“制作通知单”是从“考生名册”处获得考生信息，而不是写入，因此该数据流是错误的。

【试题二】

答案：

【问题 1】

QR：一对多。　　　QS：一对多。　　　RS：一对多。

【问题 2】

(1) C。

(2) D。

【问题 3】

层次模型采用树型结构表示数据与数据间的联系。在层次模型中，每一个节点表示记录类型(实体)，记录之间的联系用节点之间的连线表示，并且根节点以外的其他节点有且仅有一个双亲节点。

层次模型不能直接表示多对多联系，若要表示多对多的联系，可采用以下两种方法。

- 冗余节点法——两个实体的多对多的联系转换为两个一对多的联系。该方法的优点是节点清晰，允许节点改变存储位置，缺点是需要额外的存储空间，有潜在的数据不一致性。
- 虚拟节点分解法——将冗余节点转换为虚拟节点，虚拟节点是一个指引元，指向所代替的节点。该方法的优点是减少对存储空间的浪费，避免数据不一致性，缺点是改变存储位置可能引起虚拟节点中指针的修改。

解析：

【问题 1】

关系模型中实体间的联系有一对一、一对多和多对多。一对一是指一个实体只与另一个实体相联系，一对多是指一个实体与多个实体相联系，多对多是指多个实体与多个实体间的联系。

在这个系统中，一个病人只在一个病区，一个病区有多个病人，因此联系 QR 是一对多联系；一个病人只有一个主治医生，一个医生显然可以医治多名病人，因此联系 QS 是一对多联系；一名医生只属于一个病区，一个病区有多名医生，故联系 QS 是一对多联系。

【问题 2】

基本的关系代数包括并、差、广义笛卡儿积、投影、选择，其他运算可以通过基本的关系运算导出。

(1) 并(Union)。关系 R 与 S 具有相同的关系模式，即 R 与 S 的结构相同，关系 R 与 S

的并由属于 R 或属于 S 的元组构成的集合组成，记作 $R \cup S$，其形式定义如下：$R \cup S = \{t \mid t \in R \vee t \in S\}$，式中 t 为元组变量。

(2) 差(Difference)。关系 R 与 S 具有相同的关系模式，关系 R 与 S 的差由属于 R 但不属于 S 的元组构成的集合组成，记作 $R-S$，其形式定义如下：$R-S = \{t \mid t \in R \vee t \notin S\}$。

(3) 广义笛卡儿积(Extended Cartesian Product)。两个元组分别为 n 目和 m 目的关系 R 和 S 的笛卡儿积，是一个 n+m 列的元组的集合，元组的前 n 列是关系 R 的一个元组，后 m 列是关系 S 的一个元组，记作 R×S，其形式定义如下：$R \times S = \{t \mid t = <t^n, t^m> \wedge t^n \in R \wedge t^m \in S\}$。如果 R 和 S 中有相同的属性名，那么可在属性名前加关系名作为限定，以示区别。若 R 有 k1 个元组，S 有 k2 个元组，则 R 和 S 的广义笛卡儿积有 k1×k2 个元组。

(4) 投影(Projection)。投影运算是从关系垂直方向进行运算的，在关系 R 中选择出若干属性列 A 组成新的关系，记作 $\pi_A(R)$，其形式定义如下：$\pi_A(R) = \{t[A] \mid t \in R\}$。

(5) 选择(Selection)。选择运算是从关系的水平方向进行运算的，从关系 R 中选择满足给定条件的诸元组，记做 $\sigma_F(R)$，其形式定义如下：$\sigma_F(R) = \{t \mid t \in R \wedge F(t) = \mathrm{True}\}$。其中，F 中的运算对象是属性名(或列的序号)或常数，运算符是算术比较符(<、≤、>、≥、=、≠)和逻辑运算符(∧、∨、¬)。

在此主要涉及投影和选择，根据语义，(1)中“外科”与“内科”应为或关系，且应先选择再投影，因为作投影运算之后，选择操作涉及的列已经不在了，故为 C；(2)中“内科”和“胃病”应为与关系，同样应该先选择再投影，故为 D。

【问题 3】略。

【试题三】

答案：

【问题 1】

(A) 用户。　　(B) 发短信。　　(C) 管理电话簿。

【问题 2】

(A) 按数字键。　(B) 连接基站。　(C) 按断线键。　(D) 断开连接。

解析：

【问题 1】

图 7-58 给出了系统用例图，用例图展现了一组用例、参与者及其之间的关系。

易知参与者(A)是“用户”。

仔细分析，缺少的用例为“发短信”和“管理电话簿”，而“发短信”与“无线电网络”相关，故用例(B)为“发短信”，用例(C)为“管理电话簿”。

【问题 2】

根据题意，打电话的流程如下。

(1) 用户拨电话号码，每按下一个数字键，显示屏上显示相应数字。

(2) 按 OK 键进行连线，显示屏上显示“连线中……”，请求连接基站，基站通过移动电话网络连接到对方手机，若有误则返回相关信息。

(3) 接通后，显示屏显示“连线成功”。

(4) 打电话结束后，按 Cancel 键送出断线信号，通知移动电话基站断线，基站切断连

接，显示屏显示“断线成功”。

对比可得，(A)为“按数字键”，(B)为“连接基站”，(C)为“按断线键”，(D)为“断开连接”。

【试题四】

答案：

【问题 1】

状态 1：运行态。　状态 2：就绪态。　状态 3：等待态。

【问题 2】

运行进程最多 1 个，最少 0 个；就绪进程最多 N−1 个，最少 0 个；等待进程最多 N 个，最少 0 个。

【问题 3】

P2(1)、P5(2)、P1(4)、P3(2)、P5(3)、P4(1)、P1(6)，括号内的数字表示该进程还需的执行时间。

解析：

【问题 1】

进程在生命消亡前处于且仅处于 3 种基本状态之一。运行态(Running)：进程占有 CPU，并在 CPU 上运行。就绪态(Ready)：一个进程已经具备运行条件，但由于无 CPU 暂时不能运行的状态(当调度给其 CPU 时，立即可以运行)。等待态(Blocked)：进程因等待某种事件的发生而暂时不能运行的状态，即使 CPU 空闲，该进程也不可运行。

根据这段论述，易于判断状态 1 为运行态；状态 2 为就绪态；状态 3 为等待态。

【问题 2】

问题 1 给出了 3 种状态的具体表现形式。对于单 CPU 系统，运行的进程最多只有 1 个，最少可以是 0 个(当所有进程都处于等待态时)。就绪进程最多只可能有 N−1 个，因为有就绪进程的话，肯定有运行进程，最少 0 个(所有进程都处于等待态)。等待进程最多可有 N 个，最少可为 0 个(1 个运行，N−1 个就绪)。

【问题 3】

根据题意，开始调度前，各个级别队列如下。

- 优先数 1：P2(1)，时间片为 1 单位。
- 优先数 2：P5(5)，时间片为 2 单位。
- 优先数 3：P1(10)、P3(2)，时间片为 4 单位。
- 优先数 4：P4(1)，时间片为 8 单位。

注：括号内的数字表示该进程还需的执行时间。

根据调度策略“系统总是先调度级别较高的队列中的进程，仅当该队列为空时才去调度下一级队列中的进程；当执行进程用完其时间片时，它便被剥夺并进入下一级就绪队列”可知，系统先调度 P2 进程，执行 1 单位时间，时间片到，P2 亦执行完毕，此时各个级别队列如下。

- 优先数 1：时间片为 1 单位。
- 优先数 2：P5(5)，时间片为 2 单位。

- 优先数 3：P1(10)、P3(2)，时间片为 4 单位。
- 优先数 4：P4(1)，时间片为 8 单位。

系统调度 P5 进程，执行 2 单位时间，进程 P5 还需 3 单位时间，进入优先数 3 队列，各个级别队列如下。

- 优先数 1：时间片为 1 单位。
- 优先数 2：时间片为 2 单位。
- 优先数 3：P1(10)、P3(2)、P5(3)，时间片为 4 单位。
- 优先数 4：P4(1)，时间片为 8 单位。

系统调度 P1 进程，执行 4 单位时间，进程 P1 还需 6 单位时间，进入优先数 4 队列；继续调度 P3 进程，执行 2 单位时间，进程 P3 执行完毕；调度进程 P5，执行 3 单位时间，执行完毕，各个级别队列如下。

- 优先数 1：时间片为 1 单位。
- 优先数 2：时间片为 2 单位。
- 优先数 3：时间片为 4 单位。
- 优先数 4：P4(1)、P1(6)，时间片为 8 单位。

系统调度 P4 进程，执行 1 单位时间，进程 P4 执行完毕；继续调度 P1 进程，执行 6 单位时间，进程 P1 执行完毕。

至此，可得 5 个进程的 CPU 占用序列以及其占用时间为：P2(1)、P5(2)、P1(4)、P3(2)、P5(3)、P4(1)、P1(6)。

【试题五】

答案：

(1) distance(pd[i], pd[j])。　(2) !isCircuit(r, i, j)。　(3) selected(r, i, j)。

(4) h == n−1。　(5) exchange(dr, m, b)。

解析：

本题主要是函数调用的问题。

空(1)处是计算各边的长度，根据函数的声明及说明，可得应填 distance(pd[i], pd[j])。

由注释可见空(2)处是判断边(i, j)加入 r 后是否形成回路，若形成了回路，不加入，由语句 dist += dr[k].x 可知此处是将边加入，故此处应该是不形成回路条件。参照 isCircuit 函数的声明及说明可知，若形成回路返回 0，故空(2)处应填!isCircuit(r, i, j)。

空(3)处是将边(i, j)加入到 r 中，参照 selected 函数的声明及说明可得空(3)处应填 selected(r, i, j)。

由注释可见空(4)处是最后一条边条件，变量 h 表示的是“选入端点关系表中的边数”，而 n 个节点回路应该包含 n 条边，这点也可从后面 h == n 输出解看出，故空(4)处填 h == n−1。

空(5)处是进行调整，调用 exchange 函数，正确调用形式为 exchange(dr, m, b)。

7.2.8　样卷八解析

软件设计师样卷八答案与解析

【试题一】

答案：

【问题 1】

采购订单。

【问题 2】

起点：配件库存。终点：确定顾客订单。

起点：配件库存。终点：制作销售及库存情况表。

提货单。起点：更新库存。终点：顾客。

到货通知。起点：采购。终点：缺到货对照。

【问题 3】

采购单。起点：按供应商汇总。终点：供应商。

采购请求。起点：销售。终点：计算配件增量。

解析：

【问题 1】

分层数据流图中，只涉及单个加工的文件不必画出，可在子图中再画。

依此标准，图 7-63 中文件“采购订单”只与加工采购有关，故不必画出。

【问题 2】

画分层数据流图时应时刻牢记父图与子图的平衡原则。对这种数据流缺失题目，认真对照父图与子图就可得出答案。另外，还要注意与文件的交互，包括错误数据流大多也是出在此。

根据题述，图 7-64(a)是加工 1 的子图，加工 1 在图 7-63 中，认真对照其输入/输出数据流，可发现缺失数据流“提货单”和“到货通知”，进一步确定数据流的起点和终点。“提货单”是输出数据流，起点应为加工“更新库存”，其终点自然是“顾客”；“到货通知”是输入数据流，终点应为加工“缺到货对照”，起点应为加工“采购”。

另外，确定顾客订单时，需要检查库存配件，因此应有文件“配件库存”到加工 1.2 的数据流。同理，也应有文件“配件库存”到加工 1.4 的数据流。

【问题 3】

同问题 2 的分析，仔细对照父图与子图的输入/输出数据流，并确认与文件相关的数据流。

【试题二】

答案：

【问题 1】

(1) PRIMARY KEY(SNo)。

(2) PRIMARY KEY(Cno)。

(3) PRIMARY KEY(SNo, CNo)。

(4) FOREIGN KEY(SNo) REFERENCES Student(SNo)。

(5) FOREIGN KEY(CNo) REFERENCES Course(CNo)。

【问题 2】

(6) AS。

(7) Student.SNo = Grade.SNo。

(8) Course.CName = Teach.CName。

(9) WITH CHECK OPTION。

【问题 3】

(10) COUNT(Grade.CNo)。

(11) Student.SNo。

解析:

【问题 1】

空(1)处应该是完整性约束条件。在此为声明主键，据题述 Student 表的主键应该是 SNo，故空(1)处应填 PRIMARY KEY(SNo)。同理，Course 表的主键为 CNo，故空(2)处应填 PRIMARY KEY(CNo)，Grade 表的主键为(Sno,CNo)，故空(3)处应填 PRIMARY KEY(Sno,CNo)。

在 Grade 表的主键(Sno,CNo)中，SNo 是 Student 表的主键，CNo 是 Course 表的主键，这样，两者就是 Grade 表的外键，空(4)处和空(5)处是用来声明外键的，分别填 FOREIGN KEY(SNo) REFERENCES Student(SNo)和 FOREIGN KEY(CNo) REFERENCES Course(CNo)，顺序可以颠倒。

【问题 2】

创建视图的语句格式为:

```
CREATE VIEW 视图名(列表名)
    AS SELECT 查询子句
    [WITH CHECK OPTION]
```

易得空(6)处应填 AS。

为了“进行修改、插入操作时保证该视图只有计算机系的学生”，需要声明为 WITH CHECK OPTION，此即空(9)处的内容。

【问题 3】

该 SQL 语句是用于查询“每个学生的选修课程数、总成绩、平均成绩”的，显然空(10)处应为课程数，对应集函数应为 COUNT(Grade.CNo)，应该按照学号分组，故空(11)处应填 Student.Sno。

【试题三】

答案:

【问题 1】

(A) 客户。 (B) 金融中心。 (C) 提款。 (D) 转账。

【问题 2】

(A) 金额是否足够。 (B) 银行卡无效。(C) 单据打印。

解析：

【问题 1】

图 7-66 给出了系统用例图，用例图展现了一组用例、参与者及其之间的关系。

易知参与者(A)是“客户”，参与者(B)为“金融中心”。

用例“快捷提款”是“提款”的扩展，因此用例(C)是“提款”；用例“代缴费”是“转账”的扩展，因此用例(D)是“转账”。

【问题 2】

取款时，若金额不足，自然取款失败，因此判定(A)是判断“金额是否足够”。

当银行卡验证失败，服务结束，ATM 转入“空闲”状态，故(B)是“银行卡无效”。

状态(C)为“单据打印”。

【试题四】

答案：

(1) a[sc++] = dd。

(2) [a[j]][a[k]]。

(3) dist[j] >= 0 && g[k][j] == 1。

(4) −dist[k] + 1。

(5) k < 0 ?−1 : j−1。

解析：

程序中 buildG()函数的功能是输入车站数、公交线路数，以及各公交线路的车站等信息，然后构建有向图的邻接矩阵。对每一条线路，按从始发站至终点站的顺序输入线路上的车站编号。空(1)处所在的 while 循环正是实现这一功能，线路各站编号存于数组 a[]中，并用 sc 记录该线路车站总数。函数将输入的站号先存于 dd 变量中，若 dd 变量的值为−1，表示该条线路输入结束；若 dd 超出车站编号范围，则不存储；若是正确的车站编号，则将车站编号存储在 a[sc]中，并让 sc 加 1(注意这一点)。故空(1)处应填 a[sc++] = dd。

一条线路上的各车站编号输入后，接着要做的就是利用该线路的全部车站编号构建有关该线路的邻接矩阵。对于给定的该线路上的第 k 个车站 a[k]，对所有该线路上的始发站 a[0]至该线路上的第 k−1 个车站 a[k−1]的各站 a[j]均可到达 a[k]，故空(2)处应填[a[j]][a[k]]。需要特别注意单向性，即 a[j]与 a[k]次序不能颠倒；如果不是单向，则图 G 就是一个无向图，其邻接矩阵是对称矩阵。

函数 minLen()利用上述构建的邻接矩阵，求出从站 0 出发乘公交车到站 n−1 的最少换车次数。函数采用求两点之间最短路径的方法，先将邻接矩阵的第 0 行内容复制到数组 dist[]中，并置 dist[0]为 1。这样，就在 dist[]中预置了能从站 0 出发直接到达的车站。接着是一个循环，每次循环作以下事情：利用数组 dist[]，找出下一个最少上车次数的站号。如果没有这样的站号(站 0 不可达站 n−1)，或下一个最少上车次数的站就是 n−1(找到解)，则结束循环。若找到下一最少上车次数的车站，但还不是 n−1 号站，则设置该站已求得站 0 到达该站所需最少上车次数 dist[k]，将 dist[k]的值变为负值。值为负就表示已为站 k 求得解，到达站 k 的最少上车次数为−dist[k]。由于已求得站 k 最少上车次数，那些还未求得最少上车次数、经过 k 站可以达到的车站的上车次数就应作相应调整。顺序考查各站 j(站 0 除外)，若站 j 还未求得解(dist[j] > 0)，并且经站 k 能直接到达站 j(g[k][j] = 1)，并且或从站 0 不能到达

站 j，或到达站 j 的上车次数比经过站 k 到达的次数要多(dits[j] == 0 || -dist[k]+1 < dist[j])，则到达站 j 的最少上车次数改为-dist[k]+1。故空(3)处应填 dist[j] >= 0 && g[k][j] == 1，空(4)处应填-dist[k]+1。

求解循环结束有两种情况：或是没有找到下一个最少上车次数的站(k < 0)，或是下一个最少上车次数的站就是 n-1 号站。若是前者，函数因为找到解而返回-1(任意负值均可)；若是后者，从站 0 到站 n-1 上车次数为 dist[n-1]，即换车次数是 dist[n-1]-1。故空(5)处应填 k < 0 ?-1 : j-1。

【试题五】

答案：

(1) m < k。 (2) i <= twom。 (3) twoml + 1。

(4) a[i + 1][j-1]。 (5) a[i][j]。

解析：

题中已经说明该算法采用的是分治法，“就是从其中一半选手(2^{m-1}位)的比赛日程导出全体 2^m 选手的比赛日程”，再根据注释，不难确定空(1)处应填 m < k。

以下各空的判断略有难度，可通过举例法，亦即模拟执行，来生成 4 位选手(k 为 2)的比赛日程。

空(2)处应填当前问题规模中行下标的最大值，分析得当前规模行下标最大为 twom，对应 4 位选手为 4，故空(2)处应填 i <= twom。

空(3)处填日程表右上角第 1 行第 1 列元素，对应 4 位选手为 a[1][2]，此值为 3，即 twoml+1，故空(3)处应填 twoml + 1。

比赛日程中，若 A 与 B 比赛，那么 B 也与 A 比赛，它们是统一的，存在参照性。注释已说明是前一列，故列号应为 j-1；至于行号，通过分析 4 位选手比赛日程，可总结出应为 i+1。故空(4)处应填 a[i + 1][j-1]。

同空(4)处，空(5)处应填 a[i][j]。

【试题六】

答案：

(1) private。 (2) USTax* USTax::instance。

(3) instance == NULL。 (4) new USTax。 (5) instance。

解析：

在这里希望 USTax 类只有一个实例，从而需要将其构造函数设置为 private，以防止外部对这个类进行直接实例化。故空(1)处应填 private。

空(2)处是对类 USTax 的静态成员变量 instance 的初始化，应填 USTax* USTax::instance。

由于程序采用 Double-Checked-Locking 模式，即双检查锁定模式，因此空(3)处应该再次检查。也可以想象多线程的执行情况，线程 A 和线程 B 同时调用 getInstance()方法。故空(3)处应填 instance == NULL。

空(4)处是创建一个 USTax 实例，应填 new USTax。

空(5)处返回引用，应填 instance。

【试题七】

答案：

(1) private。　(2) synchronized。　(3) new USTax()。
(4) doSync()。　(5) instance。

解析：

在这里希望 USTax 类只有一个实例，从而需要将其构造函数设置为 private，以防止外部对这个类进行直接实例化。故空(1)处应填 private。

方法 doSync 是进行同步控制，因此应声明为 synchronized，即空(2)处应填 synchronized。

空(3)处是创建一个 USTax 实例，应填 new USTax()。

空(4)处要进行同步控制，故此处应调用同步函数 doSync()，故空(4)处应填 doSync()。

空(5)处返回引用，应填 instance。

参 考 文 献

[1] 全国计算机技术与软件专业技术资格(水平)考试办公室编．软件设计师教程[M]．5 版．北京：清华大学出版社，2018．

[2] 全国计算机技术与软件专业技术资格(水平)考试办公室编．软件设计师考试同步辅导[M]．北京：清华大学出版社，2013．

[3] 全国计算机技术与软件专业技术资格(水平)考试办公室编．软件设计师考试大纲与培训指南(2009 版)[M]．北京：清华大学出版社，2009．

[4] 全国计算机技术与软件专业技术资格(水平)考试办公室编．软件设计师 2009 至 2016 年试题分析与解答[M]．北京：清华大学出版社，2017．

[5] 严蔚敏，吴伟民．数据结构(C 语言版)[M]．北京：清华大学出版社，2011．

[6] [美]维斯．数据结构与算法分析(C 语言描述)[M]．北京：机械工业出版社，2004．

[7] 谢希仁．计算机网络[M]．7 版．北京：电子工业出版社，2017．

[8] 张海藩．软件工程导论[M]．6 版．北京：清华大学出版社，2013．

[9] [美]沙赫．软件工程面向对象和传统的方法[M]．北京：机械工业出版社，2011．

[10] 孙卫琴．Java 面向对象编程[M]．2 版．北京：电子工业出版社，2017．